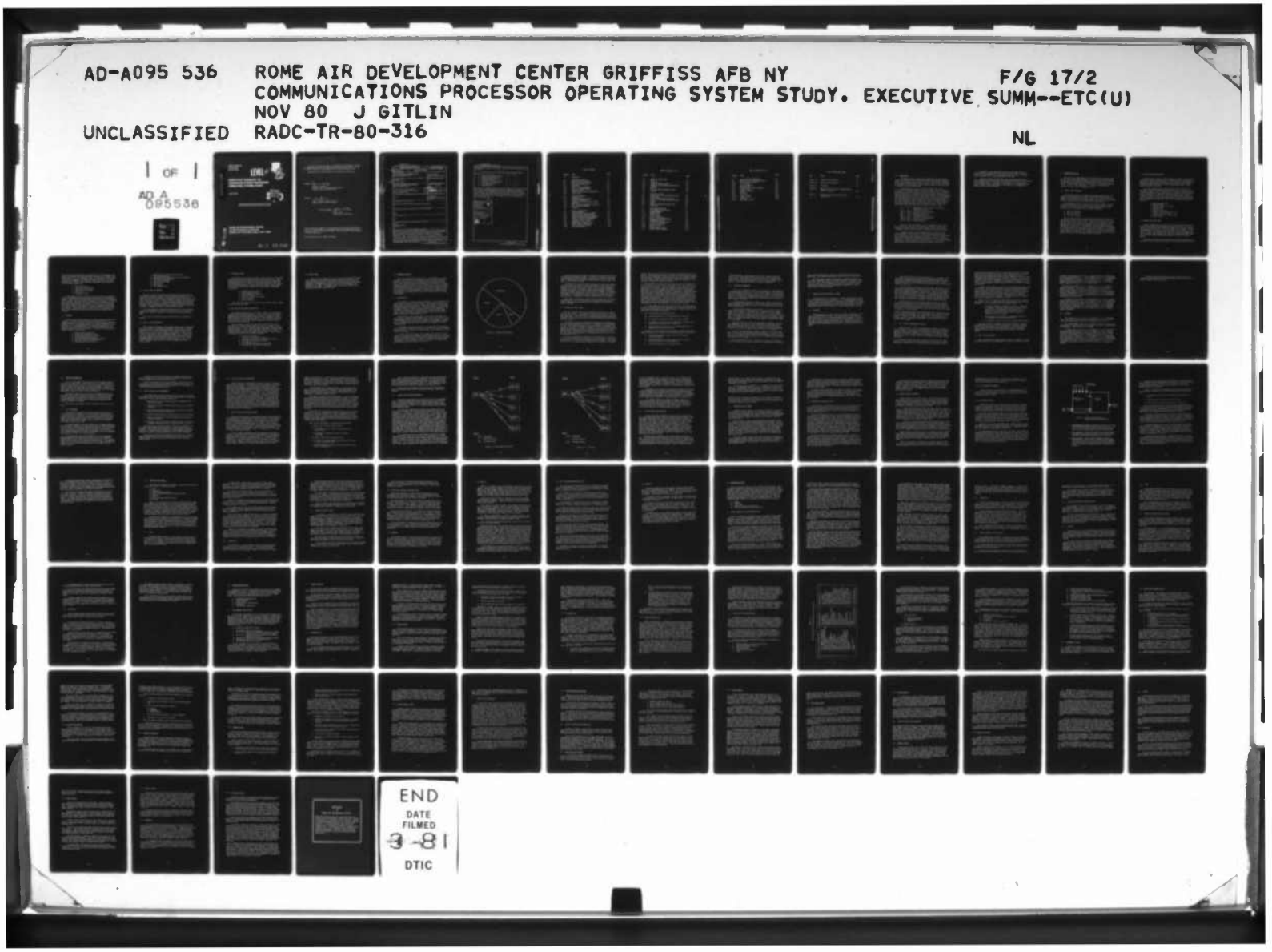
AD-A095 536 ROME AIR DEVELOPMENT CENTER GRIFFISS AFB NY F/G 17/2
COMMUNICATIONS PROCESSOR OPERATING SYSTEM STUDY. EXECUTIVE SUMM--ETC(U)
NOV 80 J GITLIN
UNCLASSIFIED RADC-TR-80-316 NL




A
955

AD A095536

RADC-TR-80-316
In-House Report
November 1980

LEVEL II



EXECUTIVE SUMMARY OF COMMUNICATIONS PROCESSOR OPERATING SYSTEM STUDY

Julian Gitlin

DTIC
ELECTE
FEB 26 1981
S D
E

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

DTIC FILE COPY

81 2 26 044

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-80-316 has been reviewed and is approved for publication.

APPROVED: Richard A. Northrup

RICHARD A. NORTHRUP
Assistant Chief, Telecommunications Branch
Communications & Control Division

APPROVED: Fred I. Diamond

FRED I. DIAMOND, Technical Director
Communications & Control Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (DCLT), Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Oata Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-80-316
2. GOVT ACCESSION NO.: AD-A095 536
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): EXECUTIVE SUMMARY OF COMMUNICATIONS PROCESSOR OPERATING SYSTEM Study: Executive Summary
5. TYPE OF REPORT & PERIOD COVERED: In-House Report
6. PERFORMING ORG. REPORT NUMBER: N/A

7. AUTHOR(s): Julian Gitlin
8. CONTRACT OR GRANT NUMBER(s): N/A

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Rome Air Development Center (DCLT), Griffiss AFB NY 13441
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 33126F, 20220001, 17 99

11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (DCLT), Griffiss AFB NY 13441
12. REPORT DATE: Nov 1980
13. NUMBER OF PAGES: 86

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same
15. SECURITY CLASS. (of this report): UNCLASSIFIED
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Switch
Operating system
Software

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report is an executive summary prepared by the RADC Program Manager for the Communications Processing Operating System Program accomplished by Plessey Fairfield and Data Industries for RADC under Contract F30602-76-C-0456. The CPOS final report consists of 9 volumes which include the major technical areas of concern in designing a secure, accountable and releable operating system that would control the hardware/software resources of an integrated switching node for the Defense

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Oata Entered)

309050

Communications System in the 1990's. The CPOS final report consists of 9 volumes:

1. Communications Switch Operating System Study Requirements Analysis — B051 853L
2. Software Reliability Study — B051 856L
3. Security Considerations Study
4. Operating System Survey — AD-B051 856L
5. Candidate Selection — AD-B051 859L
6. Implementation Methods Study — AD-B051 858L
7. Verification and Validation — AD-B051 859L
8. Design Specification — B051 860L
9. Experimentation. — AD-B051 861L

This executive summary condenses the large amount of material generated, and gives a detailed summary explanation and comments on Volumes 1 - 7. which comprise the Communications Processor Operating System Report. The information contained herein covers the type of security, accountability, reliability, and verifiability needed to control resources of an integrated communications node composed of circuit, message and packet switching.

Accession For	
NTIS GRA&I	X
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	

TABLE OF CONTENTS

Section	Title	Page
1.0	INTRODUCTION	1-1
2.0	REQUIREMENTS ANALYSIS	2-1
2.1	CIRCUIT SWITCH PROCESSING	2-1
2.2	STORE AND FORWARD PROCESSING	2-2
2.3	PACKET SWITCH PROCESSING	2-2
2.4	CLASSMARKS	2-3
2.5	CLASS II USER REQUIREMENTS	2-4
2.6	CLASS III USER REQUIREMENTS	2-4
2.7	TECHNICAL CONTROL	2-5
2.8	UDS APPLICATION SOFTWARE REQUIREMENTS	2-5
2.9	TRAFFIC MODEL	2-6
3.0	SOFTWARE RELIABILITY	3-1
3.1	ERROR ANALYSIS	3-1
3.2	SOFTWARE RELIABILITY MODELS	3-3
3.3	STRUCTURED PROGRAMMING	3-5
3.4	LANGUAGE DESIGN FOR RELIABLE SOFTWARE	3-6
3.5	MICROCODE	3-6
3.6	FAULT TOLERANT PROGRAMMING TECHNIQUES	3-7
3.7	SMALL PROTECTION DOMAINS	3-8
3.8	INTEGRITY	3-9
4.0	SECURITY CONSIDERATION	4-1
4.1	USER ENVIRONMENT	4-1
4.2	UNIFIED DIGITAL SWITCH ENVIRONMENT	4-2
4.3	CPS ARCHITECTURE CONSIDERATIONS	4-3
4.4	SECURITY KERNEL PROTECTION MECHANISM	4-3
4.5	CAPABILITIES PROTECTION MECHANISM	4-5
4.6	KEY-LOCK PROTECTION TECHNIQUES	4-8
4.7	SEGMENTED VIRTUAL STORAGE	4-9
4.8	ENCRYPTION OF SENSITIVE FILES	4-10
4.9	MEMORY RESIDUE ELIMINATION	4-11
4.10	KEY DISTRIBUTION TECHNIQUES	4-12
4.11	USER IDENTIFICATION	4-12

TABLE OF CONTENTS (Cont.)

Section	Title	Page
5.0	OPERATING SYSTEM SURVEY	5-1
5.1	HYDRA	5-1
5.2	SECURE UNIX	5-2
5.3	ESD/MITRE SECURITY KERNEL	5-3
5.4	SYSTEM 250 RECOVERABLE OPERATING SYSTEM	5-4
5.5	PTARMIGAN	5-4
5.6	MULTICS	5-5
5.7	BILL 1A PROCESSOR OPERATING SYSTEM	5-6
5.8	PLURIBUS	5-7
6.0	CANDIDATE SELECTION	6-1
6.1	DESIRED OPERATING SYSTEM CHARACTERISTICS	6-1
6.2	SECURE UNIX	6-4
6.3	SECURITY KERNEL FOR THE PDP-11/45	6-4
6.4	SYSTEM 250 RECOVERABLE OPERATING SYSTEM	6-4
6.5	PLURIBUS	6-5
6.6	HYDRA	6-6
6.7	TANDEM/16 GUARDIAN OPERATING SYSTEM	6-6
6.8	CONCLUSIONS	6-7
7.0	IMPLEMENTATION METHODS	7-1
7.1	DOD STANDARDS AND GUIDELINES	7-1
7.2	PROGRAM MANAGEMENT	7-2
7.3	PROGRAM DESIGN	7-2
7.3.1	Modular Design	7-3
7.3.2	Top-Down Design	7-5
7.3.3	Levels of Abstraction	7-6
7.4	DESIGN AIDS FOR IMPLEMENTATION	7-7
7.5	PROGRAMMING PRACTICES	7-11
7.5.1	Higher Order Languages (HOL)	7-12
7.5.2	Structured Programming	7-13
7.5.3	Top-Down Programming	7-14
7.5.4	Debugging Tools	7-15
7.5.5	Program Support Library	7-17
7.6	FORMAL DESIGN METHODOLOGY	7-18

TABLE OF CONTENTS (Cont.)

Section	Title	Page
8.0	VERIFICATION AND VALIDATION	8-1
8.1	SECURITY VERIFICATION METHODOLOGIES	8-1
8.1.1	Reference Monitor Method	8-1
8.1.2	S.R.I. Method	8-3
8.1.3	Bell-Burke Method	8-4
8.1.4	Denning Method	8-5
8.2	SOFTWARE VERIFICATION TECHNIQUES	8-5
8.2.1	Program Proving	8-5
8.2.2	Symbolic Execution	8-6
8.3	TESTING	8-8
8.3.1	Test Plans	8-8
8.3.2	Static Testing	8-9
8.3.3	Dynamic Testing	8-10
8.3.4	Debugging	8-10
8.3.5	Performance Testing	8-11

LIST OF FIGURES AND TABLES

No.	Title	Page
Figure 3-1	Software Cost Allocation	3-2
Figure 4-1	Sample Security Condition	4-6
Figure 4-2	*-Property	4-7
Figure 4-3	Relation of User Classes to Communications/ Computer Facilities	4-13
Table 7-1	Communication Switching System Levels of Abstraction	7-8

1.0 INTRODUCTION

The Communications Processor Operating System Study was performed by Plessey Fairfield and Data Industries for RADC under contract F30602-76-C-0456. This program is one of many in a series to meet the DCS requirement for an advanced communications posture, and due to the technical excellence of personnel in both companies, especially Robert Waxman of Data Industries, an outstanding job was accomplished.

The Communications Processor Operating System (CPOS) effort is one program of a multiple program effort whose purpose is the development of a Unified Digital Switch (UDS) for strategic communications. This switch will have the capability to perform circuit, packet and store-and-forward message switching in an integrated communication complex. The Communications Processor System (CPS) will control the switching node and will be supported by an operating system called the Communications Processor Operating System. It is the Government's intention that this operating system incorporate the latest advances in computer technology in order to achieve a high degree of reliability and security. To this end, the effort is divided into 9 tasks which include the major technology areas of concern. The 9 tasks of the CPOS program were reported in 9 separate volumes as follows:

VOL I	Task 1 - Communications Switch Operating System Study Requirements Analysis
VOL II	Task 2 - Software Reliability Study
VOL III	Task 3 - Security Considerations Study
VOL IV	Task 4 - Operating System Survey
VOL V	Task 5 - Candidate Selection
VOL VI	Task 6 - Implementation Methods Study
VOL VII	Task 7 - Verification and Validation
VOL VIII	Task 8 - Design Specification
VOL IX	Task 9 - Experimentation.

Because of the large amount of material presented in the volumes on the above tasks, this Executive Summary is presented, treating each volume as a separate chapter, to permit readers to select only that task of immediate interest and to provide a self-contained technical discussion.

The major conclusions and recommendations are contained in this executive summary. Most important are the recommendations in the area of reliability and system security. In particular, multilevel communications security conforming to DoD requirements represents a difficult problem for the CPOS and requires solutions which are on the fringe of the current technology. In addition, the need for high reliability is a cause of concern because of the inexact science of software technology. These concerns have resulted in heavy emphasis being given to Tasks 2, 3, 6 and 7.

Volume VIII is the system specification for the Communications Processor Operating System and does not require summary herein. The specification has been prepared as a stand-alone document suitable for the next stage of contractual or in-house development of the CPOS.

Similarly, Volume IX presents the results of the experimentation task and provides the listing of the secure processor simulation. An explanation of the computer program is presented in narrative form in Volume IX and is not repeated herein. The simulator is designed to run on the RADC MULTICS computer system. Further development of the simulator is recommended for the next phase of CPOS development.

2.0 REQUIREMENTS ANALYSIS

The first task of the Communications Processor Operating System (CPOS) was devoted to a requirements analysis. A switching center which performs circuit, packet and store-and-forward message switching, and which satisfies the requirements for Department of Defense (DoD) multi-level security, is unique. It was necessary, therefore, to establish a requirements baseline as the first step to CPOS development. The results of this task are reported in Volume I, and are summarized below.

2.1 CIRCUIT SWITCH PROCESSING

The best model available for determining Unified Digital Switch (UDS) circuit switch requirements is provided by the Defense Communications System's (DCS) Automatic Voice Network, AUTOVON. The functions performed in AUTOVON are similar to those in a commercial telephone switch except for the military requirement for multi-level precedence and preemption.

The requirements analysis of the circuit switching element of the UDS is divided into two parts: one explaining the computer related processes involved, and the other quantifying the traffic load expected. Circuit switch processes are divided into the following categories:

a. Local call set-up
b. Local call take-down
c. Trunk call set-up
d. Trunk call take-down.

Once a circuit switch call is set up, the circuit switch related software processes in the CPS remain dormant until called upon to cause a disconnect or to preempt a line or trunk. Therefore, it is important to quantify the number of originating calls per unit time (the number of call disconnects per unit time is equal to the number of call originations when averaged over the busy hour). Data obtained from prior work done for RADC in related areas were used to calculate the traffic statistics. The results of the traffic analysis are shown in Figures 2.6-1 through 2.6-4 of Volume I. The figures show, in graphical form, originating calls in the busy hour and Erlang loading in the busy hour as a function of the number of switch terminations. The switch sizes used range to 6000 terminations.

2.2 STORE AND FORWARD PROCESSING

Message switch (store-and-forward) networks have been in existence for a relatively long time. Because of this, procedures used for message switching are well established and documented. The two systems used to develop the UDS baseline requirements are the existing DCS Automatic Digital Network (AUTODIN) and the Tri-Department Tactical Communications Agency (TRI-TAC) network of AN/TTC-39 switches.

The store-and-forward processing section of Volume I discusses the functional requirements placed on the UDS by Class I users requiring store-and-forward service. We have assumed that this element of the UDS will conform to the procedures and protocol specified in JANAP 128 and the AUTODIN Interface document. The functions requiring processor action have been divided into message initiation, incoming processing, message storage, and outgoing processing. Traffic statistics for UDS sizing are also presented based on AUTODIN and TRI-TAC data. The traffic categories specified in Volume I are:

- o Average message length
- o Multiple address
- o Processing delay (by category)
- o Throughput rate
- o Retrieval rate
- o Message arrival rate
- o Traffic distribution by line speed
- o Holding time by line speed
- o Traffic intensity in the busy hour
- o Block length.

2.3 PACKET SWITCH PROCESSING

Volume I recommends that message traffic, including narrative and bulk traffic, be handled in the packet mode on interswitch trunks. The characteristics of the packet switch signalling protocol are discussed, including the recommended structure for call establishment, call disconnect, and call rejection packets. The packet formats for data transfer, interrupts, flow control, resets, and restarts are also described. We have recommended the use of the frame level procedures specified by the American National Standards Institute's (ANSI) Advanced Data Communication Control Procedures (ADCCP).

Development of packet switch element sizing and traffic projections result from a less firm base than that used for the store-and-forward element.

This is because less experience is available with this newly emerging technology and secure military packet switching networks such as AUTODIN II and SATIN IV are still in the development phase. It seems to us, however, that the standards, procedures, and protocol developed for AUTODIN II are applicable to the development of the UDS packet switch element and, therefore, have been used. Packet switch performance requirements are specified for the following characteristics:

- o Quality of service
- o End-to-end delivery delay
- o Probability of misdelivery
- o Backbone delivery delay
- o Error rate
- o Transmission rates.

Traffic projections for the UDS packet switching element are primarily based on the Defense Communications Agency specification for the AUTODIN II network. These figures were projected to the 1990 expected initial operation year of the UDS. Two sets of traffic projections are provided: one reflecting an optimistic ten percent per year growth rate, and the other giving a more modest three percent per year growth. We believe that the three percent per year growth is more realistic. Volume I presents the projections in a series of figures and tables providing terminations per packet switch node, number of interswitch channels, originating traffic per node, destination traffic per node, intraswitch traffic per node, and tandem traffic per node.

2.4 CLASSMARKS

The UDS is required to maintain user related classmarks to permit identification of processing features unique to a particular user line. The recommended approach is to handle classmarks by software whereby application program routines are called upon based on call requests made from or to the user line. It is recommended that CPOS be capable of accommodating 100 unique class-of-service requests to fulfill the requirements given in the proposed military standard, MIL-STD-188-141. The class-of-service marks included in Volume I are as follows:

- a. Maximum level of precedence
- b. Progressive conference privilege
- c. Preprogrammed conference privilege
- d. Broadcast conference privilege
- e. Call restricted to preprogrammed conference
- f. Subscriber instrument classification
- g. Trunk signalling classification

h. Restrictions on subscriber dialing access
i. Direct access service
j. Less essential subscribers for traffic load control
k. Automatic line group hunting
l. Call transfer privilege
m. Secure call privilege
n. Security level
o. Data service
p. Data equipment type.

2.5 CLASS II USER REQUIREMENTS

The Class II Users are defined as the operating personnel located at the switching center and can be further divided into two categories: switch supervisors, and switch attendants. Switch supervisory personnel have general control over switching center hardware and software and operate from a switch supervisor's console. Interface with the software, for most applications, is through the use of a structured command language which simplifies the supervisor's interface and protects the software from inadvertent modification. The switch supervisor functions discussed in Volume I include two activities: management functions and maintenance functions.

The switch attendant's interface with the processor system is much more limited. The attendant's functions are similar to those in a commercial network and include call extension, call monitoring, and directory assistance.

The Class II User functions requiring CPOS control are described in Volume I.

2.6 CLASS III USER REQUIREMENTS

Class III Users are defined as the network managers and are responsible for overall control, maintenance and management of the network. They perform their duties at a few designated locations remote from the switching center sites. A significant requirement on the operating system is the ability of the Class III User to remotely load, modify, or delete software in the switching center's Central Computing Complex (CCC). In addition, the CCC is responsible for generating various types of maintenance, fault, and status reports for transmission to the Class III User site. The DCS recommended packet format for network control messages is assumed. This format is compatible with that for the Tactical Communication Control Facility (TCCF).

2.7 TECHNICAL CONTROL

The statement of work defines technical control broadly to include both transmission and switching center monitoring and fault reporting. The CPS design includes two units which specifically are involved in CPS software and hardware monitoring. These units, the System Monitor Unit (SMU) and Performance Monitoring Unit (PMU), are described to determine their relationship to the operating system. A description of technical control application software is provided to determine its impact on CPOS. The modules included are:

a. Technical control executive
b. Analog parameter monitor
c. Analog parameter processing
d. Error rate monitor
e. Order wire communication handler
f. Fault isolation
g. Report generation.

The sizing of the modules and the load on the processor system is based on prior work done for RADC.

2.8 UDS APPLICATION SOFTWARE REQUIREMENTS

An important consideration in the design of CPOS is the requirements imposed by the applications software. A large scale switch, in particular, generates unique real-time demands on the design of the operating system. Greater emphasis is placed on the control and security features in comparison to the ability to do number manipulations. The system must have fast and efficient input/output and interrupt capabilities and must be able to manipulate tables effectively.

It was necessary to define the functional characteristics of the UDS application software before attempting to specify the design of CPOS. This was done in a series of functional module data sheets for each of the major UDS elements: circuit switching, packet switching, store-and-forward message switching, Class II switch supervisor support, Class II switch attendant support, and Class III network manager support. Each data sheet is structured to provide the following information:

a. A title for the module
b. A brief description of the function
c. The inputs to the module and the objects used by it
d. The causes for initiating the module
e. The frequency of use
f. The protection requirements of the module
g. Considerations applicable to CPOS design.

2.9 TRAFFIC MODEL

In order to have an idea of the size of the switching system envisioned by the UDS study, it was decided to create a model of a switch node. This model calculates the amount of shared central equipment required from the performance requirements which are specified for the switch in other sections of Volume I. The model was designed for and runs on the Honeywell 6180 MULTICS System at RADC. The computer model is described in Volume I and the results obtained are presented there.

3.0 SOFTWARE RELIABILITY

Task 2 of the CPOS effort was devoted to an investigation of software reliability with the goal of incorporating new and improved reliability techniques to reduce errors in the operating system. We were convinced early in the study that the use of conventional software generation techniques will not produce highly reliable software for the UDS and, therefore, evaluated new design, production, and testing procedures to reduce failures in the operational system. It is interesting to note, as shown in Figure 3-1, that the single largest component of software costs results from software maintenance. This is another way of saying that more money is spent fixing errors in the software after its initial release than in any phase of development.

3.1 ERROR ANALYSIS

The preparation of reliable software depends on an understanding of the type of errors that occur and their causes. Fault analysis has been used successfully for a number of years to help increase the reliability of the design and manufacturing process as it relates to hardware. A failure mode and effects analysis is usually useful in determining the predominant cause of hardware failures, but less well adapted to finding software errors. Typically, this technique will pinpoint design and manufacturing problems such as a weak power supply, a marginal integrated circuit, poor solder joints, and errors in the assembly process.

Software errors, on the other hand, are more subtle since they are generally less systematic than hardware errors. This results from the fact that programming does not lend itself to automation and, instead, is subject to the individual skills and thought processes of the programmer. Nevertheless, some attempts have been made to determine the systematic causes of software errors.

Two attempts at classifying software faults and their causes based on actual programs are reported in Volume II. In particular, one study reports on an error analysis relating to the development of the IBM operating system DOS/VS (Release 28).

The study performed indicates that software errors remaining in the overall program after each programmer has completed and tested his module or modules is a problem of sufficient magnitude to warrant the use of a system-wide organized design procedure. Figure 3-1 indicates that approximately three-quarters of the software costs are attributable to the testing and maintenance phases of the software life cycle.

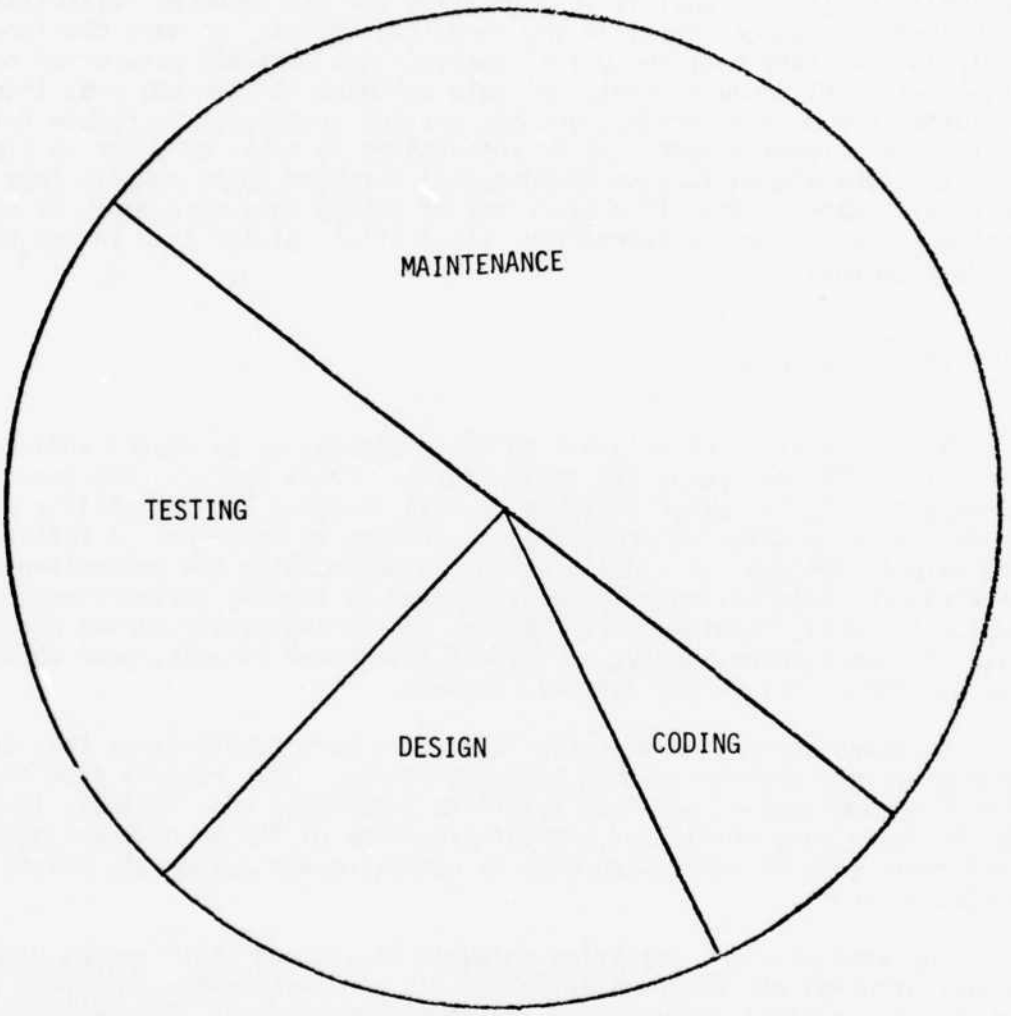


Figure 3-1. Software Cost Allocation

The main source of errors results from the more thought intensive parts of the software development process. In addition, the design, preparation, and maintenance of operating systems are more subject to errors since they exhibit a high degree of interprocess communication, serve many users, and have a relatively long life span.

Experience with previous Air Force and commercial large scale computer systems indicates that current software development processes cannot be expected to be error-free and, to the contrary, result in costly debugging throughout the life of the program. Most of these programs have been prepared on a conventional decentralized basis where programmers have been assigned responsibility for specific program parts or modules. The highest percentage of errors observed are the most insidious in that they relate to the interface of one module with another, improper communication with external devices, or incomplete or incorrect specification of the problem.

These factors lead Plessy to the conclusion that the use of an organized design technique such as top down design, structured programming and a program support library are necessary for the preparation of reliable software for the CPOS.

3.2 SOFTWARE RELIABILITY MODELS

In order to specify, monitor, and validate the reliability of complex software such as CPOS, it is necessary to develop suitable measures of software reliability and to find methods of relating these measures to objective test observations. This goal is best approached through software reliability modeling, a subject that has received a considerable amount of attention over the last five years.

In contrast to the case for software, the technology of hardware reliability is highly advanced and offers both a substantial array of mathematical tools and extensive empirical test data on component and subsystem failure rates. The subject of software reliability is far less advanced. Mathematical approaches to the analysis of software reliability are comparatively undeveloped, and sources of data to support theoretical modeling in this area are meager. Nevertheless, rapid progress is being made in this field and Task 2 undertook an exploration of methodologies which may be of use in the development of CPOS.

Of the various parameters descriptive of the reliability of a program, one of the most important is the number of errors that remain. Since in most practical cases it is impossible to demonstrate that a program is completely error free, it is necessary to accept probabilistic estimates for the number

of errors remaining, and to use these as guides in determining when to stop testing. Two other related factors of concern are the mean-time-to-failure MTTF), and the probability that the program will run for a stated time before a failure occurs. Methods of estimating all of the above parameters are useful in the development process and various models and approaches to forming these estimates are discussed in Volume II.

Plessey concludes in Volume II that the state of the art of software reliability modeling is not sufficiently advanced that any single approach can be applied with confidence. There are, however, several approaches which at modest cost can be used to provide the guidance necessary for many management decisions. It is recommended that more than one of the approaches described in the previous pages be applied simultaneously. Reasonable agreement of the results obtained by alternative methods adds confidence to the values derived by a single model, while substantial disagreement warns one that the results are not reliable and that the reasons behind the discrepancies should be investigated. In many cases this investigation will point to programming problems which may exist and suggest methods of increasing efficiency. The collection and organization of the data required to exercise these models serves to focus attention on critical areas of software testing and establishes a base of information for use in future exercises.

It is recommended that the project team selected for the development of the CPOS code be tasked with the responsibility of maintaining detailed logs of the errors encountered during system debugging. The log should as a minimum contain the following information:

(a) Time and date of detecting the error

(b) A brief description of the observed effect of the error

(c) An indication of whether the effect is minor or severe

(d) The number of testing man-hours expended and the testing time elapsed between successive error detections.

(e) The disposition of each error, and the programming time expended in the repair of each error.

The data in this log can then be used with one or more of the types of models described in this section to guide management decisions relative to:

(a) The time and manpower resources that will be needed to complete debugging.

(b) The timing appropriate to the release of code to further levels of testing, or to operational use.

(c) The final acceptance of the software product.

Plesseys opinion is that the state of the art of software reliability models is not sufficiently advanced to form the basis for formal acceptance criteria. These models do, however, provide useful management tools and their use by both the programming team and the system managers, as an adjunct to other management methods, is strongly recommended.

3.3 STRUCTURED PROGRAMMING

The use of structured programming is recommended as a good technique to increase program reliability. It is necessary, however, to mitigate some structured programming precepts in order to satisfy other Air Force requirements for a communications oriented higher order language. The Higher Order Language Investigation has studied this problem and, as a result of the anal-analysis, has recommended a communications extended version of the JOVIAL J73/I language called J73/C.

The JOVIAL J73/C language does not directly support all recommended structured programming constructs. It does, however, support the basic sequence, IFTHENELSE and DOWHILE constructs and the INCLUDE construct by means of the !COPY directive.

The structured programming CASE construct can be simulated using the JOVIAL SWITCH statement. The mechanism for accomplishing this is rather simple, the insertion of a comma after each internal controlled statement. However, the procedure is subject to programmer error. The inadvertent omission of the comma at the statement end will result in executable but incorrect code. The elimination of this hazard can be accomplished by compiler modification or by manual code review. The former approach is superior and, therefore, recommended.

Also, the capability in J73 of having multiple statements per line is not recommended. This feature violates the desirable indentation rules of structured programming and, as a result, obscures the code presentation. This problem can also be eliminated by a compiler modification or programmer directive. As in the case above, the former approach is recommended.

The use of a Program Support Library is generally considered a component part of the Top Down Structured Programming approach. We believe that this is a sufficiently desirable adjunct to good programming practice to make the technique worthwhile even if the precepts of Top Down Structured Programming are not strictly followed.

Lastly, the decision to use a precompiler to support the preparation of CPOS must depend on the computer selected for the processor system and the

higher order language chosen. Basically, the decision must rest on the availability of a suitable compiler which operates on the selected machine.

The Higher Order Language Investigation, a companion study to the CPOS effort, recommends the use of a modified version of the JOVIAL J73 dialect called J73/C. The language modification enhances the language's capability as a communications programming language and is deemed by the HOL investigation contractor to be the language most appropriate for Unified Digital Switch software development.

3.4 LANGUAGE DESIGN FOR RELIABLE SOFTWARE

The characteristics of the design of a higher order language selected for the UDS software must be a compromise. It is recommended that it be an existing language and have as many of the features found desirable in Volume II as possible. Most importantly, it should provide for abstract data types, enforcement of vertical modularity, and for strong type checking. The language should enforce the disciplines of structured programming on the user.

3.5 MICROCODE

Microprogramming is a concept which, if properly implemented, can increase the reliability of the UDS. The biggest increase in reliability comes from simplifying all of the software above the microcode. This software simplification is due to the extension of the instruction set to fit the specific needs of packet, message and circuit switching, and to the more hospitable environment created by this extension. The major reliability advantage results from simpler software which is easier to test, verify and prove.

Before the UDS can safely use microprogramming, it is necessary that more higher level languages be written for microprogram generation. The appropriate language must be oriented toward simplifying the proving of the microprogram as well as toward generating efficient microcode. Plessey believes, however, that such languages can be developed within the time frame required for the UDS.

The use of dynamic writable control store adds a new dimension to the question of microprogramming. However, because of the low level of microprogramming languages and because of the relative stability of the program mix in the operating environment, writable control store does not appear to offer any substantial benefits to the UDS. As a program development tool, it will prove extremely advantageous during microcode development and testing. Its use in the operating environment, however, may complicate the problems of security and integrity. Therefore, we must recommend against the use of writable microcode in the operational system at the present time.

Microprogramming also offers other substantial benefits to the UDS. The tailored instruction set of a microprogrammed processor will provide greater system throughput than a standard processor of the same instruction cycle time and using the same algorithm. The ability to add self-test instructions to the processor simplifies the system recovery process and speeds the repair of a failed processor.

Plessey concluded, therefore, that microprogrammability using read-only control store should be included in the UDS processors, and that efforts should be made towards the development of a suitable high-level language for its support. In addition, suitable development, test, and validation tools for microcode should be developed, simultaneously, so that they will all be available when required.

3.6 FAULT TOLERANT PROGRAMMING TECHNIQUES

Since the state of the art in programming validation is such that one can still expect errors to exist in tested software, it becomes desirable to consider building a system which will be tolerant of these errors. It has long been recognized that hardware error tolerance can be achieved through redundancy. The fault tolerant programming techniques discussed in Volume II apply the same idea to software, that is, the software modules are provided redundantly.

The primary method of software redundancy discussed involves single execution until the detection of a failure and the substitution of a redundant module. Essentially, this technique provides for the execution of a

program module followed by a specific validity test of the results obtained. A failure of the validity test results in the substitution of the alternate module. Plessey concluded that either hardware complexities will be encountered in building such a system, or high overhead will result in trying to implement some of these features in software. It appears to us that the overhead of fault-tolerant programming is unacceptable in real-time switching system, such as the UDS. Therefore, one must justify the complexity of a suitable hardware system to support failure tolerance if one were to recommend this concept.

It seems that the emphasis in the UDS is on the detection of errors, not the correction. Most of the complexity in fault-tolerant programming stems from the fact that it seeks not only to detect but to correct errors and allow the system to continue functioning. With this difference in emphasis, Plessey cannot justify the more complex schemes discussed in Volume II. They believe that there are simpler and less complex ways of detecting errors, if this is all that is required. Therefore, they cannot recommend this class of techniques for general use in the UDS.

However, there is one exception where Plessey feels that this approach may be useful. This is in the recovery and reconfiguration portion of the executive. This is an exception for the following reasons:

1. The recovery program is generally the least well tested of all programs in operating systems because of the difficulty in artificially creating hardware fault conditions. This contention is well supported by much operating system research, which shows that error recovery programs are ususally the least well tested.

2. The overhead will be tolerable in recovery since there is no requirement for very high speed recovery.

3. This program is the most critical to continued system operation.

Since the overhead is tolerable in the recovery program, a recursive cache mechanism can be used in the reconfiguration and recovery program. The recursive cache mechanism can be implemented in software without the special hardware required by the failure-tolerant parallel programming techniques and can be used in this application to ensure continuity of service.

3.7 SMALL PROTECTION DOMAINS

Plessey concluded that the small domains theory of protection is a very desirable technique for the enhancement of reliability in the UDS, and that

to take full advantage of its potential, it must be implemented both at compile time and at execution time. Plessey further concludes that an excellent way of implementing it at compile time is through the use of a language which supports abstract type extension. We also are led to the conclusion that the most effective way of implementing execution time enforcement of small domains protection is through the capabilities mechanism.

Plessey also concludes that a most important and desirable adjunct to a capability system for the enhancing of security is a technique known as rights amplification. This technique allows the direct enforcement of abstract type extension at runtime, without software intervention, as required by other schemes. Unfortunately, unlike the basic capability scheme, efficient implementations of this technique have not yet been demonstrated. The authors are hopeful, however, that such an efficient implementation will be developed in the near future.

In Volume II they recommend a design approach which enables a single enhanced capability system to implement both the desirable small domain technique of error detection and ensuring reliability, as well as providing provable military security as embodied in the ESD/MCI security kernel approach. We believe that a capability based system is a powerful means of enhancing the reliability of the CPOS and its application software and, coupled with a small domain approach using a security kernel, can provide the required security protection.

3.8 INTEGRITY

The reliability of the UDS must be maintained after it becomes operational. The Integrity section of Volume II explores the problems involved in maintaining the system's operational integrity and availability.

One of the necessary functions of the CPOS is viability testing. This testing must be included in the original design of the system and consists of two components: failure detection and recovery.

A second method of increasing system integrity is by providing redundancy, an effective but expensive method. Hardware redundancy is common. The multiprocessor configuration with dynamic load sharing offers the most reliable and cost-effective method of maintaining system throughput. The load of a single failed processor is shared among the remaining processors, with some decrease in efficiency. A fault recovery system is required, however, which assures that the essential functions are performed at the expense of background functions. Software redundancy, although appealing in its reliability, poses significant problems to cause one not to recommend this approach.

The other integrity procedures addressed in Volume II deal with methods of reconfiguring the system when a failure occurs and procedures for providing system recovery and restart.

4.0 SECURITY CONSIDERATION

Task 3 of the CPOS effort was concerned with the important topic of security for the UDS and CPS. Providing security in a computer system required to enforce multiple security levels and compartments, each of which must be isolated from one another, is a difficult problem. A number of research activities are underway which address this problem, and these have been surveyed prior to formulating their recommendations. One of the newest among these is the work of the Air Force ESD/MCI group.

An important constraint to the security considerations analysis is availability of a suitable computer architecture. Architectures proposed by various researchers are evaluated in Volume III. An analysis of the initial Communications Processor System (CPS) architecture has been performed to determine its suitability for supporting an adequate security protection mechanism for the UDS.

The Security Considerations Study is reported in Volume III of this report.

4.1 USER ENVIRONMENT

The personnel that interface with the Unified Digital Switch are classified into three categories. The first category, designated Class I, are the subscribers to the network in the traditional sense and are served by either the circuit, packet, or store-and-forward message switching subsystem. They derive communications services in terms of call connections and message exchanges and, in general, are not involved in the internal processing of the switching center.

The second category of user, designated Class II, operate and maintain the switching centers and provide assistance to the network subscribers. For the purposes of the security considerations analysis, two distinct types of Class II users must be defined since the two types differ greatly in their interactions with the UDS software and hardware. The first type are the personnel responsible for operating the switching center, and the second type are those providing Class I user assistance.

The third category of user, designated Class III, are the network managers responsible for day-to-day and longer term network management. These personnel are not located at the switching center site and, instead, perform their functions at a few designated control centers. Since they are remote from the switching node, they must receive UDS status information and exercise control by means of telecommunications lines connected to the switching centers.

In addition, Volume III also describes the special requirements of the relocatable Class I user who may derive his communication services at locations remote from his normal termination point.

Much of the guidance for the User Environment section came from the specifications for the SATIN IV and AUTODIN II network switches since it is concluded that the most pressing security problems arise in the packet and store-and-forward message switching areas.

4.2 UNIFIED DIGITAL SWITCH ENVIRONMENT

The Unified Digital Switch Environment section of Volume III contains discussions of physical and procedural security considerations in the Unified Digital Switch (UDS). Among the topics covered are:

1) The protection of user generated classified information within the switching center;

2) The physical protection of UDS software which acts on classified information;

3) The isolation of Communications Processor System (CPS) equipment, which acts on classified information;

4) Switching center personnel authorized access to classified areas;

5) Logging, journaling and alarming requirements upon detection of a security violation;

6) The types of security violations related to Class I user service, relocatable user service, Class II users and Class III users.

The future digital backbone system will consist of a network of modular switches that support circuit switching, store-and-forward message switching, and packet switching subsystems in unified centers using shared facilities. The switches will be designed to provide modular expansion of hardware and software to promote adaptability to the traffic environment.

The projected Unified Digital Switching Center consists of a Central Computing Complex (CCC), a switching network (i.e., matrix), and other peripheral equipment required for line and trunk handling, file storage and terminal access.

4.3 CPS ARCHITECTURE CONSIDERATIONS

The CPS Architecture Considerations section of Volume III analyzes the facilities available for implementing security mechanisms in the initial baseline CPS architecture. The specific areas investigated include the memory protection mechanism, the input/output structure and the interrupt structure. The philosophy behind the design of this computer complex, however, revolves about the belief that all software utilized on the system will have previously been proven to be correct. This is justified by its creators on the basis that it is intended to be used in a dedicated application, with but a single set of such programs. Therefore, they have considered only compromise due to hardware malfunction. This, we fear, may be an overly optimistic view of the state-of-the-art of program proving. We believe that some form of protection against both software errors and deliberate subversion must be provided. Analysis of the architecture indicates that a number of deficiencies exist if one attempts to implement any of the current security kernel approaches to providing protection mechanisms. Among these deficiencies are the method of memory segmentation, the assignment of security attributes to CPS units rather than processes, and a slow interrupt and trap mechanism.

4.4 SECURITY KERNEL PROTECTION MECHANISM

During the last few years, a considerable amount of research activity has been devoted to the development of security mechanisms in multiprocessing/multiprogramming computer systems. Many researchers and system designers currently believe that the security kernel approach offers the best promise of achieving the demanding goals set forth for the protection of sensitive data, including classified data in military systems. The Electronic Systems Division of the Air Force Systems Command, supported by MITRE, has been prominent in the development of the security kernel approach for multilevel classified communication systems. The experience of this group has heavily influenced our study.

In 1972, a security technology panel was formed by the Electronic Systems Division (ESD) of the Air Force Systems Command to investigate the problem of computer security in the DoD environment. This panel proposed a system which was based upon a reference monitor. The reference monitor is described as: "an abstract mechanism that controls access of subjects (active system elements) to objects (units of information) within the computer system." Subjects include active elements such as users, processes, and job streams, and objects include passive elements such as files,

programs, and peripherals. To be effective, the reference monitor must be complete, isolated and verifiable. Completeness means that the reference monitor must mediate every subject/object interaction. Isolation means that the reference monitor and its data base must be protected from unauthorized alteration. Verifiability implies that it must be small enough so that its activities can be fully verified.

The implementation of a reference monitor in a system was referred to as a security kernel which was considered to include the hardware and software mechanism required for the reference monitor abstraction. More appropriately, the security kernel is only the software mechanism when implemented in predefined computer hardware.

The system is divided into three environments: the user environment, the operating system environment, and the kernel environment. All software that is required to enforce security is limited to the kernel environment. The kernel intercedes between all subject and object interactions. With this approach, only the kernel need be verified to certify that DoD policies are being enforced with one exception, namely, trusted processes. Trusted processes are those that operate outside the kernel domain, are verified to be correct, and are not strictly monitored by the kernel as are untrusted processes.

MITRE defines four steps that are required to build a secure system beginning with a policy, development of a model, specification of the desired system based upon the model, and, finally, implementation of the specification. The policy in this case must be DoD policy, which contains two types of controls: non-discretionary controls which consist of classification levels and compartments; and discretionary controls which consist of need-to-know authorization.

There are four requirements of the model stated as follows:

- A subject shall not read an object unless:

  1) The subject's level is greater than or equal to the object's level.

  2) The subject's compartments contain the object's compartments.

- The subject shall not write an object unless:

  3) The level the subject concurrently can read is less than or equal to the object level.

  4) The compartments the subject can read are a subset of the object's compartments.

Items 1 and 2 are referred to as the simple security condition while Items 3 and 4 are referred to as the "*-property." The principles of the simple security condition are illustrated in Figure 4-1 and the principles of the "*-property are illustrated in Figure 4-2. Compartments are used as restrictions for special user communities (e.g., R and Y communities) which limit the transfer of information to members of the community only.

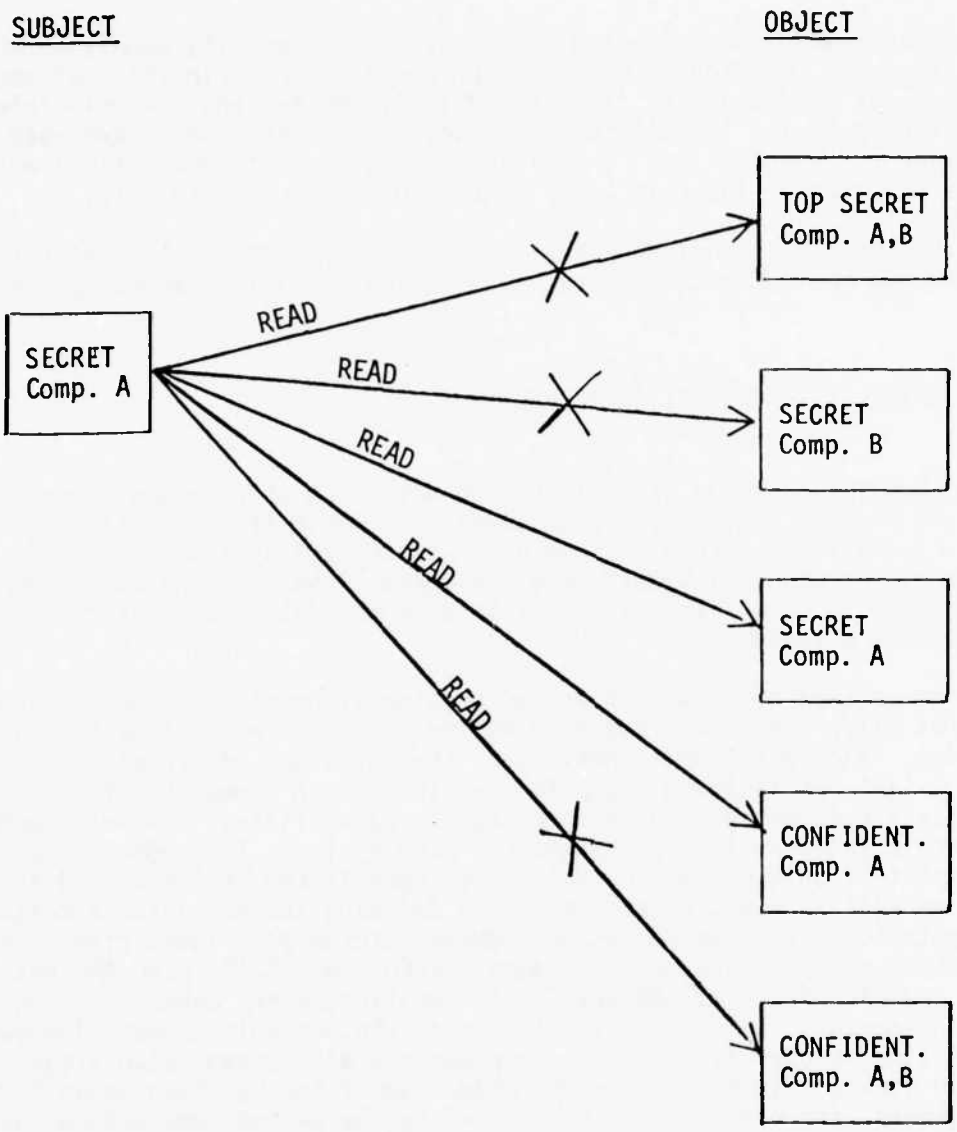
The security kernel approach and the above rules for the security model form the basis for the security protection mechanism recommended for the UDS.

4.5. CAPABILITIES PROTECTION MECHANISM

Capabilities based protection mechanisms are used in some computers such as the Plessey System 250, because of the flexibility this technique offers. Capabilities mechanisms can be characterized as systems which assign access rights to subjects (e.g., processes) before they can communicate with objects (e.g., files). This is done according to rules enforced by the operating system.

The concept of capabilities was originally proposed by Dennis and Van Horn of MIT. Dennis and Van Horn define a subject which they term a computation. A computation is defined by them as a set of processes having a common C-list, or in today's popular terminology, a common domain. Their C-list itself consists of one or more segment capabilities. Segments were the primary objects in the Dennis and Van Horn system. The segment capabilities consist of unique names by which the segments can be identified and located, as well as a set of access rights defining the permitted access of that computation with respect to the segment referenced. These rights include EXECUTE as procedure, READ as data, EXECUTE AND READ, READ AND WRITE as data, and EXECUTE, READ AND WRITE. In addition, every capability contained an ownership indicator which indicated whether this computation owned the capability in question or not. Computations with owned capabilities have broad powers with respect to the object described by those capabilities. Owned segments, for example, may be deleted by the owning computation, or access may be granted or denied other computations by the owning computation.

This granting of access is accomplished by presenting the computation receiving the granted access with a copy of the capability. At the owner's discretion, copies may have the same or fewer access rights than the original. Possession of such a copy of a capability is therefore considered proof of the owner's permission for the possessor to access the object described by it. It is for this reason that this approach is often referred to as a "ticket system"; possession of a capability for an object is considered



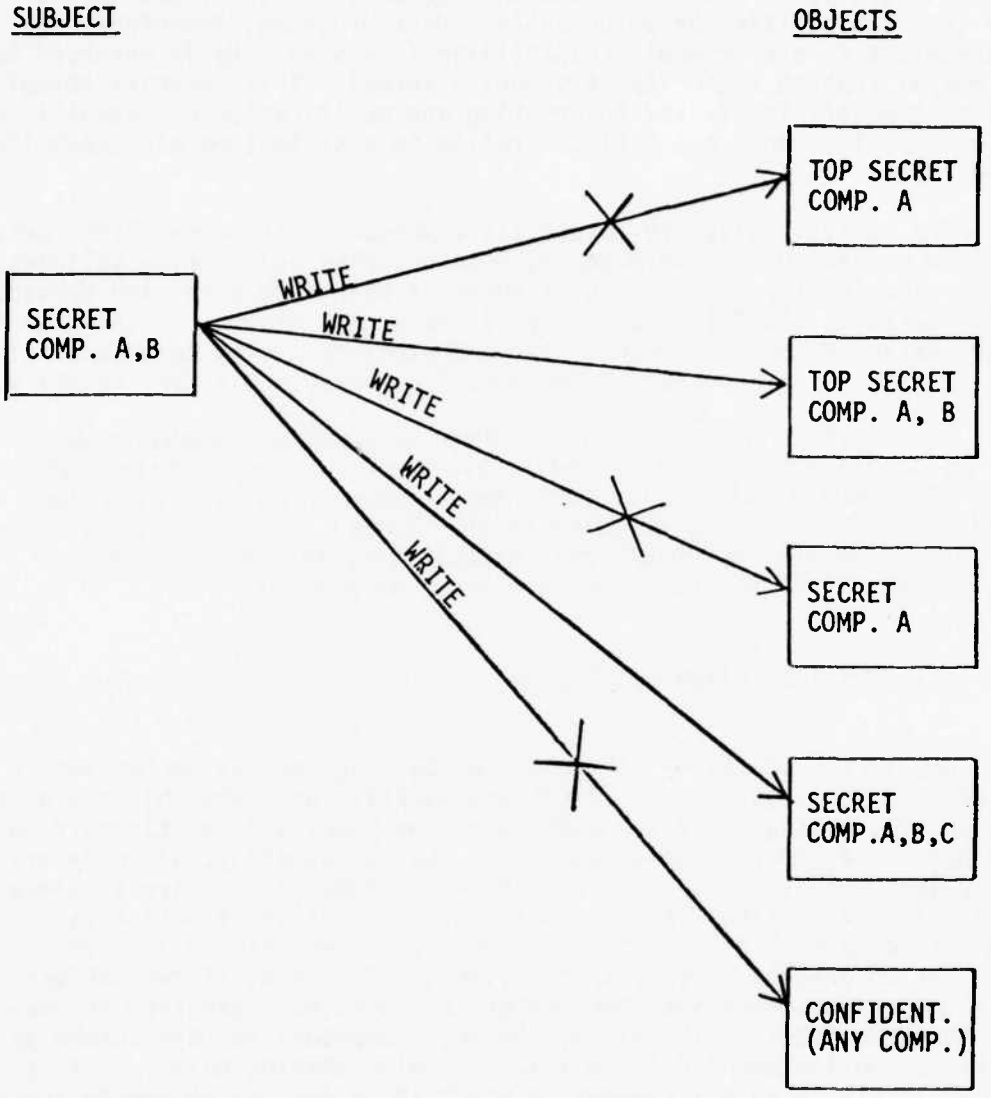
LEGEND:

Comp. = Compartment

—— = Permitted Process

⟋⟍ = Forbidden Process

Figure 4-1. Simple Security Condition



LEGEND:

Comp. = Compartment

—— = Permitted Process

 = Forbidden Process

Figure 4-2. * - Property

to be an irrefutable "ticket" to use that object. This, of course, requires that capabilities be unforgeable. User programs, therefore, may not themselves create or modify capabilities; this ability is reserved to some central trusted authority, a security kernel. This security kernel's function, however, is limited to creation and modification of capabilities, since an object's owner has full discretion in distributing his capabilities for the object.

The analysis given in Volume III concludes that capabilities provide a flexible mechanism for implementing both security and sharing policies, but that occasionally these policies conflict with each other and prevent us from realizing the full capability of one or the other. It is shown that in some instances the requirements for a protection system to ensure reliability conflict with a mechanism for ensuring provable military security.

As a result, Plessey has proposed an enhanced capabilities system which implements both the desirable small domains technique of error detection and reliability enforcement, as well as providing the provable military security embodied in the ESD/MCI kernel. The Air Force is not convinced that a conventional capabilities structure by itself can be proven secure by the techniques known at present.

4.6 KEY-LOCK PROTECTION TECHNIQUES

Key-lock protection techniques can be described as mechanisms in which each subject possesses a single key quantity and each object a single lock quantity, which, to "fit," must have some specified relationship to each other. The simplest relationship is that of equality, and many systems use just this scheme. However, if each subject is to have a unique key, then it must follow from the fact that the equality function is single-valued in both directions, that objects accessable by a given subject are accessable only by that subject. Of course, if two subjects share a common key, they are, for protection purposes, operating in precisely the same domain. Therefore, the major drawback to this scheme is obvious; it can implement only an all-or-nothing sharing policy. It is thus useful only in an environment in which it is desired to create insurmountable walls between subjects.

Another simple relationship is to permit access if, and only if, $K<L$. This relationship allows the definition of hierarchical domains, in which each domain may access its own objects and those of all domains below it (those with larger K values) in the hierarchy. Such hierarchical sharing is more flexible than the rigid wall policy discussed above, but is not sufficiently powerful for most applications, in that it generally permits

too much sharing. For example, such a system will support the military security policy insofar as levels are concerned; the imposition of compartments resulting from need-to-know restrictions, however, makes this approach insufficient for the CPOS.

From the discussion in Volume III, it is apparent that the overriding factor in a recommendation regarding key-lock techniques must be the form and degree of sharing that must be supported. In the UDS, it appears that while only limited sharing is needed, it will indeed be necessary for global data objects to be accessible by most processes.

Therefore, unless a specific, multivalued function is found which conveniently allows the implementation of military security policy, one must conclude that despite its efficiency and simplicity, key-lock techniques are not powerful enough for use by the CPOS.

4.7 SEGMENTED VIRTUAL STORAGE

Segmented virtual storage is an architectural feature of some computers which can be used as the basis for a security protection mechanism. The more advanced implementation of secure computers use memory segmentation to isolate and protect objects from subjects. In essence, virtual machines are created which have hardware and software enforced barriers between executing programs.

One of the first virtual storage mechanisms was developed in 1959 for the ATLAS computer system at Manchester University, England. The concept gained wide acceptance with the development of large scale time-sharing systems. The primary benefits of virtual storage are that programs can be written which are not constrained by the available size of main memory and the application programmer does not have to concern himself with memory management problems involved in transferrals between secondary and main memory. The virtual storage mechanism is responsible for converting the logical address entered by the programmer into the physical address where the data is stored.

Most major computer manufacturers have recognized the importance of virtual storage and offer systems containing such capability. Examples of large computers including virtual storage in their design are the Burroughs B6500, the GE 645, the IBM 360 and 370 Systems, the PDP-10 and -11, and the UNIVAC 70/46.

Segmented virtual storage systems are useful for implementing security protection mechanisms since they isolate the programmer from the machine. The logical address used by the program must be translated into a physical address by the operating system. Thus, the system must act on behalf of the program to locate stored data and, therefore, has the inherent capability of mediating these accesses.

A virtual storage mechanism can use segmenting, paging, or a combined hybrid technique for dividing memory into blocks. The investigation of security kernels protection mechanisms has shown that segmented memory is needed to support the security protection mechanism. This does not, however, preclude the use of a combined segment/page (hybrid) approach to memory organization.

4.8 ENCRYPTION OF SENSITIVE FILES

The protection of information within the UDS may be enhanced by placing the data in encrypted form prior to storage. Encryption of the data provides an additional barrier to unauthorized browsing, and protects against the accidental release of clear text information through malfunction or subversive action.

Sensitive information within the UDS may be categorized as either "user sensitive" or "network sensitive," and different protective criteria relating to each category apply. "User sensitive" information refers to traffic generated by the users and entrusted to the network for delivery to the intended recipient. NSA approved procedures for the handling of this data are mandatory, and it may be assumed that physical red/black separation, end-to-end encryption, or other protective methods as used in existing networks will apply also in the case of UDS. "Network sensitive" data refers to data such as password tables, directories, and other software that the UDS requires to perform its own mission. Regulations for the protection of network sensitive data have not been formulated to the extent applicable to user sensitive information; the proper safeguarding of such material is clearly an important function of CPOS.

For both user sensitive and network sensitive data, storage in encrypted form can be used to provide an additional layer of protection against tampering or unauthorized disclosure. However, since user sensitive data requires stringent NSA approved procedures, any additional layers of protection provided by CPOS in the form of encryption of internal files will be of less significance than in the case of network sensitive data where the additional protection represents a larger component of the overall multi-layered protection mechanism.

The design of CPOS should, therefore, consider the inclusion of cryptographic protection of sensitive data, with emphasis being placed on the protection of network sensitive data. The degree of protection to be included is properly established on the basis of engineering cost-benefit trade-offs which may, for these applications, imply a level of cryptographic protection which is considerably lower than that required under NSA approved procedures.

4.9 MEMORY RESIDUE ELIMINATION

Memory residue refers to any "image" of information that remains in memory after processing is completed and the memory cells become available for the storage of new information. Protection mechanisms must be established in the Communications Processor System to prevent the accidental or deliberate compromise of classified information that remains in either main memory or secondary storage as memory residue.

Plessey and RADC have concluded that simply writing over areas subject to reuse with a null character, in the usual manner, is a sufficient protection against compromise by the reading of "images" of classified information remaining in these areas by subsequent owners of lower classification. Plessey further concludes that the proper time for this "purging" operation to take place is at the time it is deallocated. Purging at this time allows the use of an unvalidated memory manager, as used by the CPOS System, developed at the University of Washington.

Plessey also investigated methods which may be utilized to effect this purging, and concluded that the use of purely software methods is not feasible for the UDS because of the excessive amount of processor time used and its resulting effects upon efficiency and throughput. Plessey believes that firmware implementations are feasible, inexpensive, and relatively efficient and probably represent the best overall choice at the present time.

However, where the volume of purging is sufficient to cause a difference in the required number of processors in the multiprocessing system, substituting a hardware purge peripheral for a CPU, and utilizing the DMA facility, offers the same or better performance and is more cost-effective. Under these conditions, this hardware technique becomes the method of choice.

Finally, bulk purging, while perhaps the easiest, fastest and most cost-effective technique in the long term, suffers from numerous as yet unsolved engineering problems, not the least of Which is the uncertainty of

our knowledge of the characteristics of the memory technology which will ultimately be employed in the UDS. Therefore, Plessey cannot recommend reliance upon this method at the present time.

4.10 KEY DISTRIBUTION TECHNIQUES

Advanced network security techniques are under development by the National Security Agency (NSA) which rely on key distribution procedures. This area has been investigated by Plessey for inclusion in the UDS and to determine the impact on CPOS.

4.11 USER IDENTIFICATION

Trustworthy procedures for identifying users, terminals, processes, and data are an essential element in the control of access to sensitive resources and information. The problem of providing user identification requires consideration of human factors in addition to those technologic factors that apply equally to the identification of machine resources.

In contrast with the policy common in the communication environment of authenticating identification to the level of the communications lines or facilities serviced rather than to the individual, common practice in the computer environment does extend to the authentication of individual identity. This is particularly the case in time sharing environments where a variety of password methods are usually used for this purpose.

The UDS will exist in an environment which combines aspects of both the communications and the computer environment. The relationship between the various classes of users and the communications vs computer environments is schematically represented in Figure 4-3.

The overwhelming majority of users fall into Class 1 and, following established communications policy, CPOS need take no responsibility for the authentication of individual identity for these users. There are, however, three important cases where identification to the individual (or group of individuals) level will be required. These cases are illustrated in Figure 4-1 as those users whose activities impact on the computer side of the UDS facility, either directly or indirectly, through the communications side. These three cases are defined below:

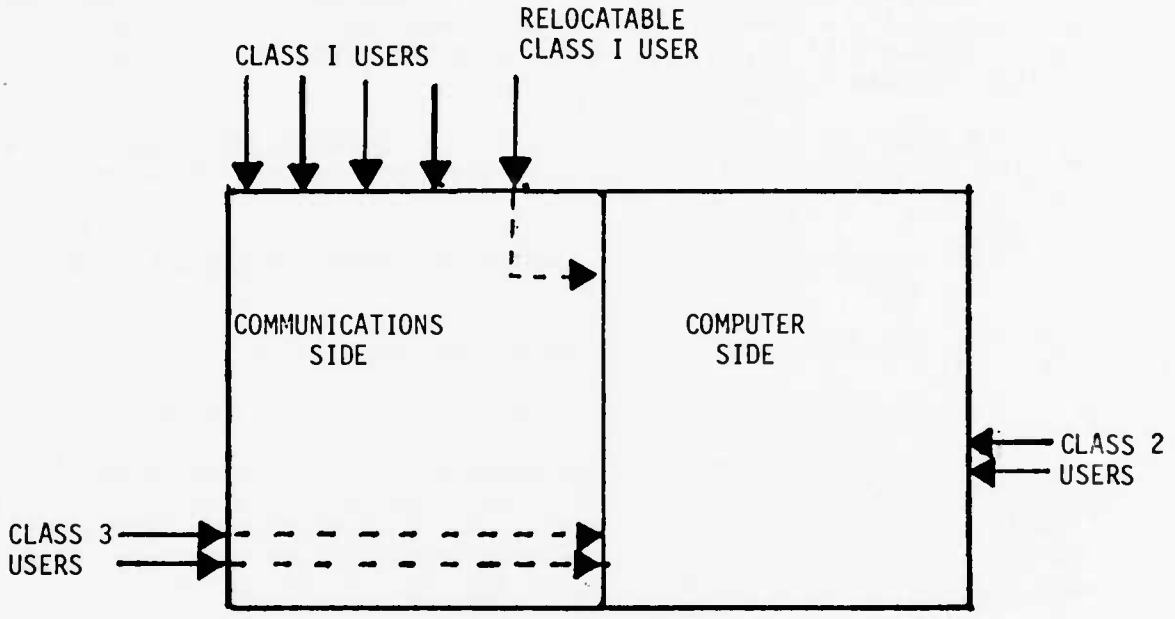


Figure 4-3. Relation of User Classes to Communications/Computer Facilities.

a) Relocatable Class I Users - Users with a given set of classmark protected services require the capability to derive this service from any compatible terminal, even though that terminal may not have previously afforded those capabilities.

b) Class II Users - These switch supervisory personnel require the capability to change directories, control traffic flow, and perform limited related operations on a local basis.

c) Class III Users - These network management personnel are responsible for overall control, maintenance and management. They require access to all data base and program information. In addition, many of these functions will be exercised remotely. These users also have the ability to downline load programs.

In order to be able to provide the capabilities listed above, means for reliable and positive identification are required. Of the three groups, the identification needs of Class II users are least stringent since a strong component of physical security can be effective in limiting the actions of these local personnel.

A number of techniques are available to perform user identification. The discussion in Volume III classifies these methods into three categories:

a) Identification by some item of information possessed by the user.

b) Identification by some object the user carries.

c) Identification by personal characteristics of the user.

Volume III evaluates each technique by determining the probability of an incorrect identification and, in addition, the probability of falsely rejecting a proper user. Although some exotic identification techniques are available, we believe that cost and reliability considerations recommend the use of the password technique described in Volume III.

The previous subsection discussed techniques for user identification. In addition, it may be required to identify terminals independently of users employing the terminal, especially where a terminal is employed at a base communications center. Techniques such as BLACKER[5] (see Subsection 7.1) are capable of combining the user identifier with the terminal identifier in a manner which permits the node to authenticate both.

The need to identify terminals in the UDS exclusively of users depends on the mode of operation. The ability of the Class II switch supervisor and Class III network manager to access Communications Processor System (CPS) software and to change programs and modify files, makes it necessary to exercise careful control over the terminals which may access the CPS. The security audit log should provide identification of the user as well as the terminal in this case.

An additional consideration for terminal identification exists if key variable distribution cryptographic techniques are used. The BLACKER system, for example, requires that the terminal to node link be secured by use of a Terminal Unique variable prior to transmission of the call variable. The Terminal Unique variable serves as a means of authenticating the terminal prior to text transmission.

An important application for terminal identification in the UDS network is the authentication of Class III users accessing CPS files and changing UDS software. As discussed in Volume I, the Class III users will be centrally located at network control facilities which are remote from the network switching sites. This requires that the sensitive function of "down-line" loading of software be accomplished via telecommunications lines which are subject to electronic surveillance and tampering.

These lines will probably be encrypted to protect access to classified files in the CPS. In addition, because of the sensitive nature of the software modification function, a method of identifying the user and terminal is necessary to provide positive identification of the remote source. It is good practice to keep a security log which identifies both the user and terminal to permit the maintenance of a security audit trail since changes to the system may be generated from multiple sources including Class II users.

5.0 OPERATING SYSTEM SURVEY

Eight operating systems were studied in the course of the operating system survey, Task 4. They are as follows:

a) HYDRA
b) Secure UNIX
c) ESD/MITRE Security Kernel
d) Plessey System 250 Recoverable Operating System
e) Ptarmigan
f) MULTICS
g) Bell 1A Processor Operating System
h) Pluribus

These can be classified by the hardware architecture used. HYDRA, Secure UNIX and the ESD/MITRE Systems use the Digital Equipment Corporation's PDP-11 family of computers. The System 250 Recoverable Operating System and the Ptarmigan system both use the Plessey Processor 250 computer. The MULTICS operating system is designed for Honeywell machines, and Pluribus is designed for Lockheed SUE computers used by ARPANET. The Bell 1A Processor operating system has been built specifically for the common control processor used by the latest classes of Bell System Electronic Switching Systems (ESS), particularly the high density ESS No. 4.

The objective of Volume IV is to document the results of the operating system survey and to determine those features of the operating systems which are useful to the CPOS and those which are not. It should be noted that the operating systems surveyed were designed for applications other than the UDS and, in some cases, for applications which did not envision communications switching. The results and conclusions developed during the operating system survey were used as the input to the Candidate Selection Task reported in Volume V.

5.1 HYDRA

The HYDRA operating system is a multiprocessor system designed at Carnegie-Mellon University for the purpose of operating systems research. It is designed to support a number of different operating systems simultaneously, as user programs. It runs on a unique set of hardware known as "C.mmp." This is a combination of commercially available equipment plus some custom-designed interfaces.

The C.mmp computer system consists of a number of small central processors connected to a number of memory modules by a nonblocking crossbar switch. All of the processors are considered as equals by the switch, although each may offer a number of different features. The central processors used are Digital Equipment Corporation PDP-11 minicomputers.

The software of the HYDRA provides a set of mechanisms which are employed by users for the purpose of building operating systems. This collection of facilities acts as the kernel of these systems. To this end, an attempt is made to separate matters of policy from the actual mechanisms which provide the service.

Plessey has concluded that to adapt HYDRA to the UDS environment would be a large task. The current process change overhead and access protection overhead is much too great for a digital switching application. Much new software would have to be written.

It is conceivable that most of the large overhead can be eliminated by adapting HYDRA to run on a capabilities-oriented multiprocessor hardware. In addition, hardware redundancy must be built into the system to increase reliability.

The security system in HYDRA is, by design, one of the most flexible to be found on any system. HYDRA provides a set of facilities by which the user can create his own security policy. Within his own operating system, the user can adopt the straight capability system provided, can override it entirely to run a completely open system, can tighten it up by the addition of levels and compartments, or even an access list policy. Some of these policies require almost no addition software, but others, such as access lists or the military-style levels and compartments, require the creation software to establish and maintain the structures.

HYDRA is just the kernel of an operating system. For the UDS application, an operating system must be designed and implemented on top of the kernel. This system must be designed to provide an environment which is especially hospitable to the real-time switching applications programs. The applications programs would also have to be written, but they would be simpler to write because of the benevolent operating system environment.

5.2 SECURE UNIX

The Secure UNIX is a general purpose multiuser timesharing system designed for use on a minicomputer system. It is designed to enforce military-style security rules on each user to protect itself and other users from failures and from purposeful attempts to compromise the system.

The minicomputers used by the Secure UNIX operating system are Digital Equipment Corporation machines. The original (non-secure) UNIX was created by Bell Laboratories for the PDP-7 and PDP-9 computers, and a later version used the PDP-11/40 and PDP-11/45, which are equipped with memory protection hardware. Versions of UNIX were also built on other members of the PDP-11 family as well as on the Interdata 8/32. The Secure UNIX is the newest incarnation and uses either the PDP-11/45 or the PDP-11/70.

The Secure UNIX is an operating system designed to provide verifiable security. It is based on a security kernel which functions as the controller of the security protection mechanism and contains only verified code. To accomplish this, the size of the kernel is kept small. The overall operating system is similar to the previous PDP-11 UNIX systems, thereby making use of extensive software developed by Bell Laboratories.

The Secure UNIX is under development at the University of California at Los Angeles for the Advanced Research Projects Agency (ARPA). Currently, it is partially operational and is undergoing testing at UCLA.

5.3 ESD/MITRE SECURITY KERNEL

The ESD/MITRE security kernel is the result of an experimental development of the Electronic Systems Division (ESD) of the Air Force and the MITRE Corporation. The computer used for the implement is the Digital Equipment Corporation's PDP-11/45. The security kernel based operating system is designed to implement the DoD multi-level security provisions. It provides a multiuser, general purpose operating environment capable of performing a number of different applications. The system is still considered experimental and development is continuing.

The system, as it is presently constituted, presents several shortcomings for use in the UDS. One of the most severe is in the Memory Management Unit (MMU). Only eight usable segment registers are provided. This results in a small address space. Another problem is that these registers must be loaded one at a time which slows down process changing.

Another deficiency is that only three different access rights are available: read-execute, read-write-execute, and no access. A larger and more flexible assortment of access rights would be desirable.

The most useful part of this system is the security protection system since it is designed to satisfy the requirements of the military security regulations in a provably secure manner.

To use this system for the UDS would require the addition of more processors to the system along with a major redesign of the memory management unit. The software would have to be rewritten to support a multiprocessor environment.

5.4 SYSTEM 250 RECOVERABLE OPERATING SYSTEM

The Plessey System 250 is a highly modular multiprocessing, multiprogramming real-time computer system. It is designed specifically for the communications switching environment, where system integrity, recoverability and maintainability are of central importance.

The system consists of a number of processors, storage units, and input/output peripheral devices, in any of a large number of possible configurations. To facilitate system integrity, a capability addressing scheme is implemented directly in the hardware to minimize the damage which may be caused by a failure. This capability addressing system involves input/output devices and the internal processor registers, as well as the memory.

The operating system consists of a collection of connected subroutines. Each subroutine has only the access capabilities that it needs, and never more. A special subset of the operating system is the programmer oriented commands of the Program Development System. It is isolated from the rest of the operating system to preserve system integrity. Also contained in the operating system is the recovery system which handles failures and reconfigures the system, as required, to bring the system back on-line.

The main advantages for considering the System 250 operating system for CPOS utilization are its use in communications switching applications and its implementation of a capabilities security architecture. Major modifications would still be required, however, to adapt this system for our application.

5.5 PTARMIGAN

The Ptarmigan system has been specifically designed for the secure military switching environment. It uses a modified version of the Plessey System 250 outlined in the previous subsection and described in detail in Volume IV. The Ptarmigan Operating System is basically a subset of the System 250 operating system with changes to better adapt the system to the transportable military switching environment. The advantages and disadvantages are the same as those given in the previous subsection.

5.6 MULTICS

MULTICS is a general purpose multiprogrammed, multiprocessor operating system. It was originally developed as a tool for operating systems development and related computer research, and has grown into a powerful operating system used to support interactive computer facilities. Much of the hardware used with MULTICS was specifically designed or modified for this purpose. The software places a heavy emphasis on ease of use for the casual user, on system security, and the privacy of each user's files.

The current version of MULTICS is designed to run on the Honeywell Series 60/level 68 computer. Previous versions used the Honeywell Model 645 and Model 6180 computers. Much of the security system is currently located in the hardware.

The software is a segment oriented, virtual memory, timesharing system. The operating system is a set of privileged segments which form the center of a hardware-protected ring structure. The system is oriented toward program development, particularly in the time-sharing mode where the support of interactive terminals is essential. Batch processes and automatically started processes, however, are also supported.

To attempt to adapt MULTICS to a digital switching environment poses problems. Because MULTICS was designed as a program development system, the process overhead consumes an excessive amount of time.

All of the applications package would have to be written, including the specialized device handlers.

Although comparatively sophisticated, there are several known holes in the MULTICS security system. In the current version, there are over three hundred segments which operate within the most protected ring of the supervisor, and, therefore, of the system. This results in the supervisor and many administrative programs having far more privilege than is necessary for their respective tasks. A failure in any overprivileged routine could compromise the privacy of other users' data and, possibly, the integrity of the system. The system also lacks an efficient means for checking the correctness of the parameters passed to a subroutine of higher privilege than the calling routine. Also lacking is a provision for two mutually suspicious programs that are attached to a single process to be protected from each other; that is, the access validity within a given ring belongs to the process, not to the active segments within that process.

To adapt MULTICS to the switching environment, a new fault recovery philosophy and implementation would have to be created. It would have to be capable of automatically restoring the system to an operating configuration quickly and with an absolute minimum of disruption.

5.7 BILL 1A PROCESSOR OPERATING SYSTEM

The 1A Processor was designed explicitly for controlling a telephone circuit switching network. It is a very fast and extremely reliable machine. The hardware is highly fault-tolerant due to redundancy of components, including the central processing unit.

The software of the 1A Processor is designed for only three basic functions: switch matrix control, maintenance diagnostics, and administrative reporting. Heavy emphasis is given to reliability and to fault recovery. The software will reconfigure to exclude failed components both quickly and automatically.

The 1A Processor is capable of being used in switching environments where a large number of transactions per unit time is required. During the busy-hour, it may be called on to handle up to a half-million calls.

The operating system of the 1A Processor provides facilities for input/output, scheduling, administrative inquiry, and maintenance. These functions can execute concurrently with call processing. The emphasis is on call processing and on maintenance and fault recovery.

Adapting the 1A Processor to the UDS environment would involve some serious changes, even though it was designed to work in a commercial switching environment. The UDS not only requires circuit switching, but also message and packet switching, all with much stronger security constraints than are present in the 1A Processor.

Security is not a primary concern of the 1A Processor design. The system is constructed to run in a controlled environment under the supervision of trained and trusted maintenance personnel. Because of this and because the subscribers do not directly use the computer, security controls in the military sense are not necessary. The environment which is envisioned for the UDS requires much stricter security - both in the hardware and in the software.

Second, since the UDS environment requires packet and message switching, the switch will use on-line storage for sensitive data. This data must be protected and the 1A Processor lacks an adequate mechanism to do so. A segmented memory with a protection mechanism, such as capabilities, is required instead of simple address bounds checking, especially when the bounds may be moved at the discretion of the programmer.

In addition, facilities are required to handle packet and message switching which have not been envisioned in the design. This represents a major modification.

5.8 PLURIBUS

The Pluribus computer is a multi-processor computer system designed for a digital telecommunications environment. It was designed by Bolt, Beranek, and Newman, Incorporated for ARPANET, the pioneering packet switching network. The Pluribus throughput is 1.5 megabaud, an improvement over the previous switch processors used at the network nodes.

The hardware consists of Lockheed built SUE processors, memories, and input/output devices interconnected by high speed busses. The architecture is described in Volume IV.

An operating system in the conventional sense does not exist in the Pluribus system. As an ARPANET node, the software consists of an interweaving of several applications programs. The major design goal of the software is to maintain the system's availability. If a failure occurs, the software attempts to isolate the failed component, remove it, and continue processing.

The ARPANET nodes are not designed to be secure in the military sense. No control is exercised to prevent the interconnection of terminals and computers other than those imposed by individual users. The ARPANET nodes are designed with the assumption that the system is operating in a benign environment. As a result, an ARPANET node is more prone to disruption by a malicious user than would be acceptable in a UDS node since a rigid security protection system is absent from the design.

6.0 CANDIDATE SELECTION

Volume V presents the results of the Candidate Selection Study, Task 5. The study considered the changes required to convert five operating system candidates selected from the previous operating system survey task. A sixth operating system, the Tandem/16 Guardian Operating System, was also included since the architectural features are attractive for consideration for CPOS. Unfortunately, the authors were unaware of this relatively new system at the time the operating system survey was being performed, but it was considered important enough to include in the candidate selection. The five candidates chosen from Task 4, in addition to the Tandem/16 system, are:

a) HYDRA
b) PLURIBUS
c) Secure UNIX
d) Security kernel for the PDP-11/45
e) System 250 Recoverable Operating System.

6.1 DESIRED OPERATING SYSTEM CHARACTERISTICS

Volume V evaluates each candidate system on the basis of the following features: security, memory management, process management, input/output management, interprocess communication, reliability, and human interface. These, of course, are not mutually exclusive groupings; there can be considerable overlap between several, depending upon the design of any given system. The CPOS must contain procedures to implement each of these areas.

One of the most stringent requirements of the CPOS is imposed by the need for multilevel security for handling classified messages. This facility is a machine implementation of the people-and-paper document classification system consisting of several different subject groupings, each containing several levels of sensitivity. These subject groupings are known as "compartments." One compartment could be NATO, another Bioweapons, Nuclear, USSR, and so forth. The classified levels applicable are the standard DoD security levels: unclassified, confidential, secret, and top secret. These are ordered by successive restriction: if a person has clearance to access secret material, then related unclassified and confidential material is also accessible, but material classified as top secret or above is not.

In the area of process management, it is clear that UDS applications processes can be divided into at least two classes. These are operations control processes and non-operations control processes. This latter classification includes management reports, directory assistance and the like.

The operations control processes must take precedence over this second group. Each of these groups can be subdivided repeatedly by various criteria to form a priority list of tasks required by the priority scheduler.

Some sort of memory management algorithm will be required. It is highly improbable that the entire UDS application software and the CPOS can be contained in main memory in a static fashion. Even if this were possible, some memory management would be required for allocating data space. It is apparent that several different types of memory allocation will be required for the UDS application software. A memory-resident allocation will be required for the commonly utilized operations control programs and for the data tables which will be accessed by these programs. For non-operations programs and for diagnostics, a secondary storage-resident allocation is more practical. For call registers, message buffers, and other data structures, still another type of allocation will be required. Much of the information utilized by the application software for routing and security will be of a tabular form. This is most easily dealt with in the form of data bases.

Input/output management is intricately mixed with the security structure. Some devices will have a fixed security level; others, especially remote devices, may have a changeable security level, depending upon the person using it. Assigning devices to processes must also be considered; care must be taken to prevent processes from halting because of conflicts for resources.

Then there is the question of what types of input/output devices must be supported by the CPOS. Certain ones are obvious: a console terminal, disk drives or other form of secondary storage, and some type of magnetic tape drive. For operations, the CPOS must support control of a circuit switching matrix, devices for sending and receiving data packets, and perhaps remote keyboard terminals and data encryption hardware.

Since the UDS will include a store-and-forward message switching capability, some secondary storage space will be required for material which cannot be delivered immediately. This requires a file system. The security structure will probably require that different security levels, and perhaps even different security compartments, be stored upon physically distinct media. A separate secondary storage device for each security level will probably be sufficient. However, it is possible that some messages will contain several parts, each with a separate level of classification. The file system must successfully deal with this. The best approach appears to be a multilevel tree structure, with a separate major branch for each security level and each security level branching into a number of security compartments.

In any multiprocessing computer system, some means of communicating between processes is a requirement. The crudest methods utilize shared files or common tables. More sophisticated techniques resemble standard input/output to the applications programmer. In a multicomputer, such as is envisioned for the UDS, data may frequently have to be transferred between processes executing on separate processors. A strong and sophisticated interprocess communication system can provide implicit solutions to data locking and processor synchronization problems. For example, if a single process is given the task of providing all services involving a given data base, the problems of reading partially updated data, unlocking locked records, and invalid access, all have simple solutions because all requests for data base service are given to the one "gateway" process.

Another basic requirement of the CPOS is reliability. Reliability not only includes correctness of code, but availability. When a failure occurs, either in the hardware or the software, the system must handle the error in a manner which allows the system to function and remain available. It cannot be allowed to breach security, in any case. To provide this reliability requires several things: fault detection, automatic system recovery, and fault diagnosis.

Fault detection is not only a hardware function, but also includes software. The classical fault is a device failure indication reported in a hardware status register. These include memory parity, I/O device error, and illegal instruction. The software must detect faults such as garbled data in a table, parameters out of range, or array index out of bounds. Some faults can be detected by the appearance of an exceptional condition; others must be ferreted out by special tests.

Once the fault has been detected, the CPOS must recover from the error and continue processing. This recovery must be quick and automatic. In some cases, fault correction may simply consist of retrying the operation that failed. In more severe cases, major parts of the system may have to be excluded from the system for diagnosis and repair in the case of the hardware, or for modification in the case of software. How this process commences is determined by the structure of the hardware: a multiprocessor offers the system designer many more options than a prime-and-standby system. This translates into more flexibility, especially in regard to recovery speeds: many faults will only disrupt one or two processors of a multiprocessor while the remainder of the system continues normal operations.

Once the fault has been detected and the system has recovered, time must be taken to determine the cause of the fault. This requires that the CPOS be capable of running diagnostic tests concurrently with normal operations. Diagnostic tests can be divided into two groups: fault detection

and fault isolation. There will be hardware components in the CPOS whose sole function is to detect faults. Some fault detection diagnostics must be scheduled to test CPS mechanisms on a periodic basis to detect "quiet" failures. The remainder can be scheduled on demand, either as a result of a detected failure, or as the result of a request from the operations personnel.

6.2 SECURE UNIX

The use of the Secure UNIX as a basis for the CPOS has several advantages, but also some disadvantages. The main advantage is that applications programs can be written and tested on any standard UNIX system in a higher level language and then ported to the CPOS. This allows the parallel development and implementation of the CPOS and the application programs.

The disadvantages of the Secure UNIX are substantial. The only resemblance which a CPOS UNIX would have to the Secure UNIX is the basic multikernel structure and the standard UNIX virtual machine seen by the application routines. The security kernel has to be redesigned in order to implement the levels and compartments security structure and to allow for multiprocessing. The file system kernel and the initiator will also require substantial redesign. A system recovery strategy, which cannot breach security itself or allow the recovered system to breach security, must be designed and implemented.

From an operations point of view, a CPOS UNIX would probably suffer from extensive process change overhead due to the large number of small domains. This may prove to be detrimental to the performance of the system.

6.3 SECURITY KERNEL FOR THE PDP-11/45

The ESD/MITRE security kernel is a monolithic structure designed for a single processor machine. Most of the problems which would be encountered in the implementation of this system are due to the architectural differences between the PDP-11/45 and the, as yet to be determined, CPS machine.

The major advantage of this system is the security structure which is already designed into the system and which conforms almost specifically to the needs of the CPOS environment.

What may prove to be the major disadvantage is the requirement to certify a relatively large quantity of code for trusted processes which do

not conform to the security rules. At the current state of the art, it is unlikely that so large a quantity of code can be rigorously proven.

The ESD/MITRE Security Kernel is designed to be an experimental demonstration system. Because of this, operational concerns such as efficiency and speed are of secondary importance. It is, therefore, very likely that a good number of the algorithms used in the design of the Security Kernel will have to be redesigned before they can be utilized in the CPOS.

6.4 SYSTEM 250 RECOVERABLE OPERATING SYSTEM

The advantages of the Plessey System 250 are that it is not only a multiprocessor system, but that it is also known to work. This system has seen extensive field experience, and was designed for use in communications switching. It already has most of the features required for the CPOS built into it, including the fault recovery system.

The biggest disadvantage of the System 250 is that it requires modification to the capabilities mechanism to provide the required security protection mechanism. To add the security structure requires significant redesign of all interprocess communication and all input/output. This also affects the fault recovery strategy, in order to prevent breaches in security during the recovery process.

6.5 PLURIBUS

The advantage in using the Pluribus design for CPOS is that it already performs one of the major tasks of the UDS, namely, packet switching. It is also a multiprocessor system and, as a result, contains the system software needed for controlling multiple processors and providing resource allocation among processors. The fault recovery system is also well developed.

The disadvantages of using the Pluribus software as the CPOS are, however, serious. Primarily, the security protection structure is not adequate for the UDS since no separation is provided between the operating system and the application code, and there is no security policy enforcement. Among its other disadvantages are its lack of higher level languages and its lack of a file storage facility. These shortcomings for the CPOS application require substantial redesign.

6.6 HYDRA

An advantage of HYDRA for use as the basis of CPOS is that it supports a flexible multiprocessing structure. The interlocks and other problems associated with multiprocessing and parallel processing have, for the most part, already been solved. The use of a capability protection structure simplifies the problems of maintaining system security. Most importantly, HYDRA has been written in a higher level language and is extensively documented; making the porting of this system to a more suitable hardware environment comparatively easy.

On the disadvantage side, the lack of the required security structure and file structure loom as major problems. HYDRA was designed to be the kernel of a number of different operating systems simultaneously, not to be an operating system in and of itself. Because of this, most of the other features required by the CPOS, including the command language and the fault detection and fault recovery system, must be written.

6.7 TANDEM/16 GUARDIAN OPERATING SYSTEM

One of the newer commercially available computer systems is the Tandem/16. This system, designed and developed by Tandem Computers, Incorporated, utilizes a unique combination of hardware and software to produce a high speed, reliable, and flexible multicomputer. It is used for transaction processing in applications where system reliability is of primary importance.

The processor has a stack oriented architecture and utilizes a sixteen bit word length. It is totally microprogrammed, with the instruction set tailored specifically for the operating environment. A processor is capable of addressing four stacks containing a maximum of sixty-four thousand words each. Two of these stacks are the code stack and data stack of the operating system. The remainder are the code stack and data stack of the user processes. The code stacks cannot be modified. The input/output devices are all two-port devices; these ports are normally connected to separate processors.

The Guardian operating system forms the heart of the Tandem/16 computer system and is designed to provide a reliable fail-soft environment with automatic fault recovery. In this, it appears to succeed admirably, and, in the process, provides workable solutions to the difficult problems associated with multiprocessing. Guardian is a flexible, general purpose multicomputer operating system which provides for both program development and specific applications.

A disadvantage of this system is the lack of a multilevel security protection mechanism adequate to meet UDS requirements.

It is the opinion of the authors that this security can be provided by redesigning the interprocess communication facility of Guardian. Different levels of security can be stored on physically separate storage devices, each of which is driven by a separate, but identical, driving process.

The major advantage of the Tandem/16 computer system is that it already provides working solutions for two of the most difficult problems of the CPOS task: support of a real-time multiprocessor environment and automatic fault recovery. An additional advantage is that the Tandem/16 is designed for on-line transaction processing, a task which is similar to that of the CPOS.

6.8 CONCLUSIONS

Of the operating systems examined during the course of this study, none is directly suitable for use as the CPOS. In fact, not one can be made suitable without extensive redesign and modification.

Of the features considered desirable in the CPOS, some are far easier to design into an operating system than are others. The easiest to provide are driver programs for various peripheral devices. The two most difficult are the multilevel security feature and the interprocessor communication in a multiprocessor computer architecture. The CPOS must provide both of these features.

Including either of these features in an operating system are difficult problems. Many attempts have been made to write viable multiprocessor systems, and some work only marginally. A few attempts have been made to implement multilevel security in an operating system and none have been demonstrated in an operational communication switching environment.

Because the difficult problem of multiprocessor interconnection is a solved problem for a limited number of operating systems, the authors recommend one of these candidates be used as a base for designing the CPOS. With this selection, the biggest design task remaining is to add multilevel security. If one of the experimental secure systems is selected as the model for the CPOS, there is a risk that the design will be unworkable in the UDS switching application.

The Tandem/16 operating system, Guardian, has desirable reliability and multiprocessor handling features, but lacks a multilevel security structure. This deficiency is serious, and requires significant modification to the existing commercial version of Guardian. The user interface does provide, however, almost all of the features which the CPOS will be required to provide, and does so in the most consistent and logical form of all of the operating systems studied.

Since the Guardian operating system serves a multiprocessor computer architecture and since it incorporates required reliability features for a real-time transaction oriented computer system, we believe that this operating system can serve as a design model for the CPOS.

7.0 IMPLEMENTATION METHODS

Volume VI of the final report provides the results of the Implementation Methods Study, Task 6. The objective of this study was the defining of an approach for producing the CPOS which satisfies the requirements for reliability and security as determined under Task 2 and Task 3, respectively. The major subject areas discussed in Volume VI are:

a. DoD standards and guidelines
b. Program management
c. Program design
d. Design aids for implementation
e. Programming
f. Formal design methodology.

7.1 DOD STANDARDS AND GUIDELINES

The applicable DoD standards are briefly described in Volume VI and the sections of the standards which are most applicable to the CPOS implementation are noted. Some of these standards are general in nature and are not affected by the specific program development techniques which are recommended for the CPOS. Other standards require some modifications, particularly with respect to their requirements for conventional flowcharts as documentation for software systems. Where appropriate, recommended revision is stated.

The standards reviewed are as follows:

a. MIL-STD-490, Specification Practices
b. MIL-STD-483, Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs
c. MIL-STD-187, Switching Planning Standards for the Defense Communications System
d. DoD 4120.17M, Automated Data Documentation Standards Manual
e. DoD 5000.31, Interim List of DoD Approved HOLs
f. DoD 5200.28M, ADP Security Manual.

Plessey recommends that the standards discussed in Volume VI, appropriately annotated, be used for the CPOS implementation. It may also be appropriate that new standards be developed to cover the use of new and evolving software development technologies such as top-down and structured programming. The RADC Structured Programming Series is a good starting point for the development of these standards.

7.2 PROGRAM MANAGEMENT

The discussion of program management techniques provided in Volume VI is divided into two parts, one discussing the management organization for a software project, and the second discussing program control methods.

The main management organization methods reviewed in Volume VI are the chief programmer team and egoless programming, and variations of these techniques which go by the names of Strong Leader, Strong Leader plus Board of Directors, Inspector General, Councils and Working Group, and Project-Oriented Blind.

The program control methods reviewed are the familiar Critical Path Methods (CPM), of which Program Evaluation and Review Technique (PERT) is the most popular, design and coding reviews, and software audit and review.

While we recommend the use of a systematic program management methodology such as those discussed in Volume VI, we believe that any particular management procedure may work well in one organization and fail in another. Success or failure could be due to the diversity of skills of the members of the task, the underutilization or overburdening of the task managers, the size of the project and its expected duration, and a particular company's experience and commitment. For these reasons, we do not recommend that the Government make any management organization approach a contractual requirement for the generation or implementation of CPOS. With respect to program control methods, it is recommended that the institution of some form of critical path method such as PERT be a requirement for any major software development. Various design reviews and system audits of the types described in this section should also be encouraged but not be mandatory.

7.3 PROGRAM DESIGN

The program design phase of the implementation process is vitally important to the assurance of reliable and cost effective software. Although software development techniques are in their infancy, certain methods have worked successfully for the development of programs of similar size and complexity to CPOS. The foremost among these methods is the use of modular design.

It has been generally accepted by software designers, that dividing a program into a number of self-contained modules is superior to large integrated complex programs from the viewpoint of reliability, readability

and maintainability. The reasons for this become evident to anyone attempting to review a program prepared by someone else. The methods for good module construction, however, are, at best, inexact.

What is required are criteria for judging good module design, from both the characteristics of the module's internal construction and from its interface with the other modules comprising the overall program. Internal construction is discussed in Volume VI under the subject heading "Module Strength," while the interface criteria are discussed under the heading "Module Coupling." Also, included is a systematic procedure for dividing a program into modules, called "Composite Analysis." This decomposition of the program is presented in a step-by-step manner.

A successfully employed design methodology which has been used with modular programs is Top-Down design. This technique imposes a design and implementation procedure which requires an orderly progression from the more general to the more specific, in an organized hierarchy of stages. The concept of top-down design is also discussed in the program design section of Volume VI.

The program design section includes the Levels of Abstraction design method first introduced by Dijkstra. A design structure for CPOS is developed based on a concept outlined by Shankar and Chandersekaran and the software approaches recommended in Volume I on Requirements, Volume II on Reliability, and Volume III on Security.

7.3.1 Modular Design

The concept of dividing a complex system into simpler modules is useful for the CPOS implementation, since it leads to software that is more reliable and easier to maintain. The basic problem with the modular design approach, however, is that too many small modules can cause processing inefficiency.

Using the guidelines of module strength and coupling given in Volume VI can result in a large number of small modules if improperly applied. Although small modules are desirable from the viewpoint of reliability since they are clearly defined, they pose an efficiency problem because of the interconnection complexity.

In general, the interconnection and passing of parameters and data between modules requires greater overhead than when the same operations are performed within a module. The CPOS will operate in a real-time environment; consequently, consideration must be given to minimizing

processing overhead by trading between the greater reliability of smaller modules and the increased efficiency of larger modules.

The problem still remaining is how to design larger modules without seriously degrading reliability. Based upon our analysis an effective way to form larger multifunction modules is as follows:

1) Decompose the CPOS into elementary functions that exhibit high module strength and low module coupling.

2) Identify high-usage of functions.

3) Group related high-usage functions together to form modules.

Decomposing a complex system into larger modules containing related functions results in a slight problem for software maintenance. Namely, if a single function within a module requires replacement, the entire module must be dealt with. This is not a serious problem, because the structure of a module is less complex than the overall program and the number of lines of code affected is limited.

Another efficiency consideration which results from module size has to do with the security protection mechanism. Volume III, dealing with the Security Considerations Study explains the security benefits to be derived from computer architectures using segmented memory. In this case, variable length named data and instruction blocks are stored and protected under the control of the security protection mechanism. The easiest and generally best procedure to follow is to treat modules as system segments.

Establishing a one-to-one correspondence between modules and segments, however, can lead to efficiency problems. The use of many small modules increases the complexity of segment "bookkeeping" structures such as access control lists and various capability tables. Modules that are too large have the potential of causing security problems in that the protection system is not sufficiently fine-grained to provide adequate flexibility.

From the above discussion one sees that the job of dividing a program into modules is not straightforward. The program designer must balance the often contradictory criteria of efficiency, reliability and security. Also, the most desirable method of assigning functions to modules is sometimes not obvious.

Software designers have attempted to provide qualitative guidelines for judging the "goodness" of modules by giving criteria for determining

module cohesiveness and coupling. These criteria, discussed under the topics of Module Strength and Module Coupling in Volume VI, do not, however, provide a methodology for generating good modules but, rather, a non-exact way of judging quality after the module has been generated. Composite Analysis, however, is an attempt to provide a procedure for designing modules in a top-down manner with the aid of block diagram layouts based on the detailing of the functions of the program. This technique is explained in Volume VI.

These procedures must be treated as non-exact. They provide general guidelines for the designer to follow, but still require a large measure of experience and judgment on the part of the program designer. As a result, the specification of a formal procedure for generating CPOS modules does not appear feasible. The use of a modular structure for CPOS, however, is recommended because of the significant reliability benefits to be gained.

7.3.2 Top-Down Design

As discussed in Section 2.1 of Volume II on software reliability, the two most common causes of software errors are an incomplete or inaccurate understanding of a program's function (i.e., by system implementors) and logical errors in the construction of programs. Traditionally, these types of errors have been difficult to detect during the implementation phase (i.e., design and coding) and are often not detected until later in the maintenance phase (i.e., after the system is operational). It is, therefore, highly desirable to adopt a methodology during the implementation phase which minimizes the number of errors in this category and promotes user/implementor understanding of the function of the system. Top-down design is one such methodology.

A systematic approach such as top-down design is especially important in a system such as the Unified Digital Switch, where security and reliability are critical factors. Undetected errors, especially those that compromise security, are particularly serious and implementation procedures which help to eliminate such errors are important.

The use of top-down design during the implementation phase has two major advantages as follows:

1) It promotes user/implementor understanding by reducing complex tasks into functional components. This is accomplished by refining the understanding of the task in layers of increasing

detail. Thus, users and implementors can review the design as it develops with user participation in the critical early stages.

2) It allows details to be deferred until after major components have been analyzed and direction decided. In this way, time-consuming detailed analysis does not begin until both the user and implementor are satisfied that the design reflects the system's requirements. Thus, major design flaws are encountered early in the process and rework is minimized. This feature is especially useful on Government contracts where the Government's technical personnel can review the direction of the design prior to the large scale commitment to the implementation process.

We believe that the top-down design approach is an organizational procedure which is useful during the CPOS implementation to enhance software reliability. For this reason, the authors recommend that it be used during the implementation phase of the CPOS.

7.3.3 Levels of Abstraction

Levels of abstraction is a computer system organizational technique that can be used to partition complex systems into a hierarchy in which modules at the same level bear some relationship to one another, usually in terms of the control of resources (e.g., processors, input/output channels, paging drum, data structure, etc.). The key to understanding the benefit of levels of abstraction is to understand the criteria used to partition levels. The hierarchy organization is such that the lower levels are more directly related to the hardware than the upper levels. Communications between levels are rigidly structured to permit upper levels to initiate the activity of the lower levels and not to permit the reverse procedure. For that matter, the functions of the higher levels are masked from the lower levels which perform tasks in support of the higher levels.

There are two stages of development in the levels of abstraction concept. First, partitioning of the system into levels (decomposition); second, defining the hierarchy among the levels (hierarchical structure). An effective method of performing the partitioning, as outlined by Parnas, is to identify the critical components of the system, especially those that are likely to change, and create a level around each of these components. Thus, the relationship within each level and the relationship between levels will be easily understood (promoting reliability) and adaptable to change.

The selection of partitions for the levels is critical to deriving the inherent benefits of the technique and depends on the application of the computer system. The selection process should be guided by commonality of function rather than oriented toward sequential events, as is common in "flow chart" design systems. That is, the software should be organized by selecting resources, particularly hardware, which should be hidden from other levels of the computer system. Thus, access to the resource is clearly defined, an aid to the generation and maintenance of reliable software; and more easily controlled, an aid to assuring security.

Volume VI provides a design for CPOS based on the levels of abstraction approach. The partitioning into 22 levels as presented by Shanker and Chandersekaram for a communication switch is used. A summary of levels, objects realized, and hidden details is contained in Table 7-1. The details of the UDS application functions and system functions performed at each level are contained in Volume VI.

7.4 DESIGN AIDS FOR IMPLEMENTATION

The complex task of coordinating the various phases of implementation of a large-scale program is greatly improved by the use of design aids. A design aid is any scheme, computer program, or manual procedure that assists the implementation staff in understanding and developing the computer program. Design aids can be applied at any phase of development from program initiation to its final stages. The objective of using design aids is to simplify the implementation phase and to provide more reliable code.

These aids could be pictorial or graphic or could include text. If the text is formalized, the design aid is capable of being automated. Under a structured programming environment, where rules of coding are more rigid, capability for automated design is enhanced and there is better chance for improving the final product. Nevertheless, some phases of implementation might require manual design aids.

The program design and implementation process can be divided into functional parts as follows:

a. Function Analysis and Modular Specification
b. Data Base Requirements Analysis
c. System Architecture
d. Data Architecture
e. Control Structure Analysis
f. Coding.

TABLE 7-1.

Communication Switching System Levels of Abstraction

Level No.	Levels of Abstractions	Objects Realized	Hidden Details
21	USER COMMAND INTERPRETATION	User Commands	User Protocols
20	HOST COMMAND INTERPRETATION	Host Commands	Host Protocols
19	OPERATOR COMMAND INTERPRETATION	Operator Commands	Operator Interface
18	JOURNAL BALANCING	Journal Files	Storage Release
17	CHECK POINTING	Check Point Files	System Snapshots
16	MESSAGE SYNTHESIS	Messages	Message Assembly
15	USER PROCESS MANAGEMENT	User Processes, Signals	Scheduling, IPC
14	MESSAGE VALIDATION	Validated Messages	Validation Criteria
13	MESSAGE ACCOUNTABILITY	Accounted Messages	Accountability Criteria
12	MESSAGE DISTRIBUTION	Message Queues	Distribution Criteria
11	PACKET SYNTHESIS	Packets	Packetizing Criteria
10	PACKET VALIDATION	Validated Packets	Validation Criteria
9	PACKET ACCOUNTABILITY	Accounted Packets	Accountability Criteria
8	PACKET DISTRIBUTION	Packet Queues	Distribution Criteria
7	DIRECTORY MANAGEMENT	Directories	Directory Implementation
6	VARIABLE SEGMENT MANAGEMENT	Segments	Storage Addresses
5	FIXED SEGMENT MANAGEMENT	Activated Segments	Memory + I/O Addresses
4	SCHEDULED PROCESS MANAGEMENT	Scheduled Processes	Processors, Interrupts
3	PAGE MANAGER	Pages	Paging Policies
2	FORMATTING	Formatted Messages/ Packets	Formatting Criteria
1	CHANNEL HANDLING	Lines, I/O Devices, etc.	Line Protocols, etc.
0	HARDWARE	Interrupts, Descriptors	Hardware Implementation

The first two functions listed above are not well suited to automation since they require analysis of the specifications to assure that modules and data structures conform to the specifications. A top-down programming approach enhances this process.

The third phase, that of the specification of system architecture, has features which can be computerized. In particular, if the architecture is built on a top-down design principle, the system design forms a tree structure and various mathematical algorithms are available to analyze and optimize the system structure. Also, alternate structures can be compared and contrasted. Similar possibilities exist for the development of the data architecture.

An analysis of the control structure can be aided by the use of a program design language (PDL) whose syntax is, in many ways, similar to that of a higher order language but allows for embedded English descriptive sentences. Any PDL can be considered as having two types of constructs:

1. Logical Constructs

    a. Conditional Branching
    b. Repeated Execution
    c. Sequences

2. Modular Constructs

The logical constructs can be made compatible with structured programming, and the modular constructs can be specified to finer levels of detail as the system design develops. At these later stages the full resources of automata theory could be brought to bear to develop automated design aids. Of course, a system structure must be available to the PDL program.

The minimal PDL is one which indicates flow and decision. Under structured programming, the vocabulary of a small PDL contains constructs for conditional branching, repeated execution and sequencing. The remaining information is contained in imbedded English text which is not amenable to computerized analysis.

As the system and data architecture develop in the implementation process, certain "established" modules and data are entered into the PDL as part of its vocabulary and a "higher order" PDL results which has appropriate data references and modifiers. In the final stage of the process, the modules are replaced by accurate HOL code.

A gap exists, however, in the early stages of implementation where knowledge of the system and data structure is known to a reasonable degree, but the PDL is too formal to accommodate that information. A useful tool at this stage is some form of graphic presentation such as flowcharting or HIPO. Any device which enhances the presentation of information about the system is of value.

PDL and graphics thus can be used to complement each other at different stages of the implementation process. In the early stages of the design process, graphics is of greater value, while at the later stages the formality of the program design language is more important. Even during this period, however, graphics serve a role as a means of overview or summary.

The implementation process is also aided by various tools and procedures which help in its successful completion. These include the following:

1. Compilation and debugging
2. Design and data file modification and update
3. Text editing
4. Job control
5. System descriptions and summaries.

Rather than have a ragbag of unrelated software tools, it is more desirable to group them into one comprehensive package. It is also desirable that this package be capable of modification and expansion as the need arises. This design "kit" could include automated software packages such as an automatic documentation program, or automatic software design analysis programs. Such a group of design aids, organized into one system, is a useful method of making all the tools available to the implementation team. A manual is required which states how the various tools are accessed, how the output is interpreted, the tools' limitations, and statements of precautions and caveats.

Another alternative, leading hopefully to program proving, is the use of a formal design specification language. This language, sharing the features of the language of mathematics, forces precision in design specifications and eliminates or reduces ambiguity. Further, the formal character of the language allows computerization through which verification or proof of correctness can be provided.

The design aids explained and evaluated in Volume VI are as follows:

a. Program Design Language (PDL)
b. Design Expression and Confirmation Aid (DECA)
c. Problem Statement Language (PSL)/Problem Statement Analyzer (PSA)
d. Software Requirements Engineering Methodology (SREM)
e. Higher Order Software (HOS)
f. Programmer's Workbench (PWB)
g. National Software Works (NSW)
h. Hierarchy Plus Input, Process, and Output (HIPO).

The evaluation reported on in Volume VI indicated several alternate implementation approaches and resulted in the following recommendations:

1. A free form Program Design Language offers sufficient advantages to the specification of CPOS to make its use appropriate. The Program Design Language used should incorporate the recommended structured programming constructs.

2. A program module structure analyzer should be used. A good example of such an analyzer is the Design Expression and Confirmation Aid (DECA). A number of other analyzers are discussed in this section and the selection of a particular one, however, must be influenced by the computer selected for Unified Digital Switch implementation.

3. In the event the Digital Equipment Corporation's PDP-11 is selected as the CPOS development machine, the commercial availability of the Programmer's Workbench version of the UNIX system (PWB/UNIX), offers significant advantages to the Government. The PWB/UNIX software package offers a powerful off-the-shelf development and word processing capability.

4. If contracting procedures permit, the availability of the National Software Works (NSW) has the potential of offering a large repertoire of useful implementation aids.

7.5 PROGRAMMING PRACTICES

An important component of the implementation phase leading to reliable software is the programming practices employed. Section 6 of Volume VI is devoted to this subject. The areas discussed are higher order languages, structured programming, top-down programming, program debugging, and the program support library.

7.5.1 Higher Order Languages (HOL)

Most coding for large systems is done today by means of higher order languages (HOLs). The advantage of the use of a HOL is that the coding can be accomplished with relatively minor regard to the machine on which the software will run. A good HOL will reduce machine considerations to a minimum and address itself, when necessary, to such problems by a suitable modification or restriction of the language.

This does not imply that all coding must be done in a HOL. Possibly, to improve efficiency and throughput in critical modules, these modules may have to be coded in assembly language.

The choice of the HOL to be used for CPOS is based on a number of factors including:

1. Programming experience with the language.
2. The ease of maintenance of the software for the ultimate user.
3. The compatibility of the HOL with structured programming principles.
4. The capability to effect data manipulation and data packing efficiently.
5. The production of efficient machine language code by a suitable composer.
6. The ability to verify code and prove correctness.
7. The availability of suitable compilers for target machines.

The use of structured programming is recommended as a good technique to increase program reliability. It is necessary, however, to mitigate some structured programming precepts in order to satisfy other Air Force requirements for a communications oriented higher order language. The Higher Order Language Investigation, a companion contract, has studied this problem and, as a result of the analysis, has recommended a communications extended version of the JOVIAL J73/I language called J73/C.

The JOVIAL J73/C language does not directly support all recommended structured programming constructs. It does, however, support the basic sequence, IFTHENELSE and DOWHILE constructs and the INCLUDE construct by means of the !COPY directive. The ENDIF and ENDDO delimiters must be introduced as imbedded comments in J73/C. This would not allow for diagnostic checks if the present version of the J73/C design is implemented.

The structured programming CASE construct can be simulated using the JOVIAL SWITCH statement. The mechanism for accomplishing this is rather

simple, the insertion of a comma after each internal controlled statement. However, the procedure is subject to programmer error. The inadvertent omission of the comma at the statement end will result in executable but incorrect code. The elimination of this hazard can be accomplished by compiler modification or by manual code review. The former approach is superior and, therefore, recommended.

Also, the capability in J73 of having multiple statements per line is not recommended. This feature violates the desirable indentation rules of structured programming and, as a result, obscures the code presentation. This problem can also be eliminated by a compiler modification or programmer directive. As in the case above, the former approach is recommended.

Lastly, the decision to use a precompiler to support the preparation of CPOS must depend on the computer selected for the processor system and the higher order language chosen. Basically, the decision must rest on the availability of a suitable compiler which operates on the selected machine.

If the Government decides on the use of an existing machine, the decision on the use of a precompiler depends on the availability of a suitable existing compiler for the selected machine or, alternatively, the difficulty of modifying an existing JOVIAL compiler which generates code for the target machine. The precompiler approach should be considered only if a significant cost savings can be achieved by using an available compiler.

7.5.2 Structured Programming

Plessey recommends that the use of structured programming be a contractual requirement for the implementation of CPOS because under a top-down programming environment it will enhance the implementation effect, minimize programming errors (especially of the logical type), and facilitate the production of more readable code. The last feature is due to the nestability feature of structured programming which allows the capability of indentation.

On the other hand, structured programming should be viewed at its proper programming level. The constructs of structured programming are not

necessarily to be viewed as HOL code. A rigid adherence to the "letter" of structured programming could produce unnecessarily awkward HOL code. Thus, an IFTHEN in HOL could be, in reality, a degenerate form of IFTHENELSE. From this point of view, a discussion of the merits of IFTHEN is unnecessary.

The key features required of structured programming for our application are:

1. Elimination of unconditional branches

2. Single entry and exit points for each structured programming block

3. The use of three fundamental structures:

   a) Sequence
   b) IFTHENELSE
   c) DOWHILE/DOUNTIL

4. Use of the optional structures of: CASE and INCLUDE

5. Enforcement of the nestability property.

This last property is what allows for the capability of indentation.

With regard to JOVIAL J73/C, note that most features are compatible with the structured programming requirements and those that are not can be made so by the introduction of a precompiler or a modification of its design. The latter course is recommended unless significant cost savings can be achieved by the former course.

7.5.3 Top-Down Programming

Top-down programming is a good approach for implementing modular software systems, but is not without some problems for software implementors. As indicated in Volume VI, the difficulty in using top-down programming lies in the potential complexity of the stub structures which are required to replace lower level modules which have not as yet been prepared. The stubs are needed to test control structures and to simulate data needed by the module under test.

The top-down approach, however, is recommended over more conventional procedures which attempt to integrate modules during a final integration

phase. The benefits to be gained by early integration as facilitated by top-down programming are derived from increased reliability and, therefore, less subsequent rework.

In addition to the use of the top-down approach, we recommend the use of structured programming for the benefits to be derived from increased program reliability. Recently, software implementation methods have been developed which combine the two techniques to provide an organized design and implementation procedure which provides the benefits of both.

An interesting design documentation technique has been developed for top-down structured programs called "Structured Control-flow And Top-down programming structures," which has the acronym SCAT. The major benefit of the SCAT approach is that the form of the presentation is similar to that of traditional flowcharts. Since most programmers in the field today are familiar with flowcharts, the transition to SCAT charts causes minimum difficulty.

The basic building blocks of SCAT diagrams are described in Subsection 6.3.3 of Volume VI. Also described are the procedures used to document top-down hierarchy levels and structured programming construct control transfers. The program design in SCAT form appears clear and easy to follow since it avoids the complex decision branching of standard flowcharts.

7.5.4 Debugging Tools

A variety of debugging aids have been evaluated in Volume VI that are currently available, but the majority of these aids are designed for a particular computer or higher order language. For this reason, most of these design aids cannot be directly applied to CPOS implementation in their present form, but they do contain features which are desirable.

After reviewing the various debugging aids, Plessey concluded that the CPOS implementors will need, as a minimum, a facility to perform overall and selected memory dumps. The following features will be useful in such a program.

a. An interactive dump capability for displaying the contents of memory locations at various stages during program execution.

b. A selective dump capability for displaying specific memory locations.

c. Octal or hexadecimal data presentation as well as symbolic presentation for some applications.

d. Base 10 scientific notation of memory locations containing floating point data.

Plessey has examined a variety of higher order language debugging aids and they believe that the approach exemplified by the Extendable Debugging and Monitoring System (EXDAMS) is useful for CPOS implementation. The major advantage of the EXDAMS approach is that it provides more flexibility with respect to the types of debugging information it can supply, than the other techniques examined. This results because EXDAMS creates a history tape of the entire program execution which can be queried to obtain a variety of useful debugging information.

The features considered useful for CPOS implementation and which could be provided by a debugging aid such as the EXDAMS are as follows:

a. Statement-by-statement execution tracing.

b. Summaries of the number of times each statement is executed, including the number of times each conditional branch is followed.

c. A mapping capability between variables and the statements that alter their values during program execution.

d. A listing of select variables and how their values changed during the program execution.

e. A list of non-referenced variables.

f. Diagnostics in response to program interrupts caused by software errors.

A useful feature which cannot be easily implemented with the EXDAMS approach is the ability to change the value of variables at breakpoints. This is difficult with the EXDAMS approach since the programmer interacts with a history file of his program after the program execution is completed. Thus, an additional run-time interactive debugging aid would be required to implement this capability.

Breakpoints are not necessary with an approach such as EXDAMS since a complete history tape is created for the test run. The programmer can query the system to obtain information about any point by means of the test run record.

In conclusion, good debugging tools are an important and necessary aid to the preparation of a complex program, and CPOS is no exception. The finding of subtle errors during the implementation process significantly improves operational reliability and, as a result, decreases software maintenance costs. We recommend, therefore, that the selection of the UDS development computer and the HOL consider the availability of adequate debugging tools and, if necessary, provide for the development of such tools.

7.5.5 Program Support Library

Programming Support Libraries have evolved as a mechanism to aid in the orderly development of Top-Down structured programs. The Program Support Library serves as a repository for data required to support the programming process and is useful throughout the entire development process, including the design, coding, testing, documentation and maintenance phases. In addition, a Programming Support Library is an effective mechanism for controlling and coordinating the development activities by providing a facility for monitoring progress and determining that scheduled deadlines are met.

A significant design advantage is that designers and programmers are relieved of much of the burdensome, clerical procedures associated with the project. Furthermore, coordination between project members is simplified since all data concerning the system is available in the common library, thus minimizing errors that result because of misunderstandings between project members. Other advantages of the program support library are that it is easier to shift personnel from one task to another to meet schedule deadlines, it makes training of new personnel easier, and it minimizes the effort required to perform technical audits of the system.

A Programming Support Library is a useful tool for CPOS development for both the implementation and maintenance phases. A PSL is useful to programmers since it automatically performs many tedious functions that otherwise occupy the programmer's time. A PSL is useful to management because it provides information about the current status of the project, and is especially useful for the CPOS implementation because it can provide the COTR with up-to-date information concerning the contractor's progress. Similarly, a PSL can store and manage information and data required for software maintenance. Software maintenance personnel will be able to access current as well as previous versions of all programs and files.

A specification for a Programming Support Library is contained in the RADC Structured Programming Series referenced in Volume VI. Plessey recommends that a PSL should be used by the CPOS implementors which conforms with the referenced document.

7.6 FORMAL DESIGN METHODOLOGY

A large-scale system is designed and implemented in many stages and involves the efforts of many people (including programmers and managers). A number of factors are considered and these are communicated to the design and implementation team in the form of flow charts or other graphic representations, English narrative description, PDL or HOL code, tabular summaries and other ways. The end result of the implementation process is HOL or assembly language code, user's manuals, graphic representation, and narrative description. Determining the correctness of the software which results from this process is usually a formidable job. Even if formal proof of correctness is not required, the accountability of the software for its intended purpose is still difficult to establish. The difficulty appears to be that the various "descriptions" of the system are usually incomplete or vague so that determination, let alone proof, of system correctness becomes a difficult job. A language that is precise and analyzable by the computer would clearly help if it is adequate for design description. Such a language is one which is mathematically based.

The design specifications for the CPOS system are capable of being written in the formal language of the first-order predicate calculus. Being a mathematical language, it is capable of the exactness and precision necessary for mathematical proof. The process does not, in essence, add any new elements to the design process except the use of mathematical formality and is not a panacea. It does, however, force the designer to be precise, resolve ambiguity and use a language which is amenable to computer analysis and which offers the capability of being proved correct.

Even if formal proof of correctness is abandoned, the methodology of formal design specifications can be used and verified informally as to correctness. The non-ambiguous nature of the specifications will give the informal verification a greater measure of certainty and confidence. The authors therefore recommend its use in specifying the critical sections of CPOS such as the security kernel.

8.0 VERIFICATION AND VALIDATION

Volume VII of the final report presents the results of the study of software verification and security validation techniques performed under Task 7. The base line for this study was the work performed under Task 2, Software Reliability Study, and Task 3, Security Considerations Study.

The Reliability Considerations Study contains a number of ideas and concepts which can be used to enhance the reliability of an operating system. Task 7 developed the most promising concepts pertinent to assuring the correct operation of the CPOS and resulted in recommendations specific to CPOS verification and validation.

The Security Considerations Study explored the special problems of providing multilevel security in the communications processor system environment. It detailed various methods for obtaining and maintaining the required securie software. The objective of the verification and validation study is, however, to provide a means for verifying the security of the operating system regardless of which techniques are used to establish the secure environment.

8.1 SECURITY VERIFICATION METHODOLOGIES

A number of different research teams have been investigating the question of verifying that a computer system is secure. Each of these teams has selected a model of a secure system as a starting point, very often choosing the Bell and LaPadula model.

Four of these techniques are of sufficient development to merit consideration for verifying the security system of the CPOS. Two of these techniques were developed by teams at the MITRE Corporation. One uses a security kernel approach and was developed by a team headed by J. K. Millen. The second was developed by D. E. Bell and E. L. Burke. SRI International has also developed a technique which is of interest and has much in common with the two by MITRE. The last method described takes a different approach. Proposed and developed by D. E. Denning of Purdue University, this technique was developed for a more general class of security systems, of which military-style security is a subset.

8.1.1 Reference Monitor Method

J. K. Millen has proposed and demonstrated a validation technique based on the concept of a reference monitor. This method relies on the use of formal specifications which are reduced to tables of security relationships. These relationships are then proven.

The reference monitor, or security kernel, is a level of abstraction which is immediately above the level of the hardware. It is this code which controls access to all of the objects in the system. Higher level processes send requests to the reference monitor to perform any function which might have security related aspects. Some of the functions which are provided by a reference monitor are:

a. Create or activate an object
b. Delete or deactivate an object
c. Give a subject permission to access an object
d. Remove a subject's permission to access an object
e. Change security level of a deactivated object.

This list of functions does not provide a complete set. Depending upon the exact design of the reference monitor, a number of other functions must also be added.

It is required that the reference monitor be able to protect itself. For this reason, it utilizes the privileged instruction mode of the hardware. Any other features which affect the integrity of the reference monitor or allow a higher level process to bypass the reference monitor must be reserved for the exclusive use of the reference monitor.

Some system designers feel that certain security related functions, such as initial classification of incoming data, is best handled outside of a security kernel. This creates a need for "trusted subjects" which, although they do not obey all of the axioms of the secure system, provide necessary functions which help to maintain a secure operation. Although the security properties of these trusted subjects must also be proven, they are not included in the reference monitor and must be proven independently.

In order to be able to validate the security properties of the reference monitor, this approach proposes the use of two levels of written specification. Both levels must present exactly the same virtual environment to the user; the entire difference between them rests upon the degree of detail involved. The higher level of specification describes the security properties of the reference monitor. The lower level describes the software implementation of these properties.

8.1.2 S.R.I. Method

A group of researchers at SRI International, consisting of R. J. Feiertag, K. N. Levitt, P. G. Neumann, and L. Robinson, have also been investigating the problem of validating operating system security. In their initial work, an attempt was made to avoid the use of a mathematical model of a secure system or the use of a reference monitor. This research proved not to be viable. Because of this, they adopted a modified version of the mathematical model of D. E. Bell and L. J. LaPadula.

The Bell-LaPadula security model contains a formal definition of security as explained in Volume VII. To this definition, the group at SRI has added a concept of integrity which is the dual of Bell and LaPadula's security. According to their model, security is intended to prevent the reading of information by unauthorized subjects, while integrity is intended to prevent the destruction of information by unauthorized subjects.

The security conditions require that the security level of each object be greater than or equal to the security level of the writer and be greater than or equal to the security level of any other objects whose contents were read to provide the value to be written. The dual of this, the integrity conditions, require that the integrity level of each object be less than or equal to the integrity level of the writer and be less than or equal to the integrity level of any other objects whose contents were used to derive the value to be written.

The methodology involved is based upon a five level system design process. The initial level of this process is to define the interface to the operating system as seen by the user. The second level takes these modules and subdivides them into a hierarchical structure consisting of a number of different levels similar to Dijkstra's levels of abstraction concept. The next stage creates formal specifications for each of the modules and submodules of the hierarchy. The fourth level of their system design process is the creation of mapping functions which designate the state of a module in terms of the states of modules on the lower level upon which a given module depends. The fifth and final level implements the module specifications as actual executable code.

The SRI team's validation effort is based on these specific system development stages. Each of these stages has specific properties which must be verified. On the first level, these properties are the security and integrity conditions, along with the other axioms of the secure system model. On the next level, consistency of the hierarchical structure must be shown. On the third level, each of the module specifications must be shown to be

complete and consistent. The mapping functions of the fourth level must then be shown to be consistent with the hierarchical structure and with the module specifications. At the last level the actual programs must be shown to be consistent with the mapping functions and with the module specifications.

8.1.3 Bell-Burke Method

D. E. Bell and E. L. Burke have proposed another method of validating the security properties of a software system. They introduce the concept of a validation chain which is used to link the model of an ideal secure machine to the actual implementation. The chain consists of a number of steps and to show that each adjacent steps in the chain are logically equivalent is sufficient to validate the entire chain.

The starting point of the chain is a mathematical model of a secure system. It first must be shown that the formal specification corresponds to the model, then that the software specification corresponds to the formal specification and, finally, that the usable machine corresponds to the software specification.

Bell and Burke believe that having only these steps in the chain is not always enough to make the validation process easy to accomplish and understand. They state that the length of this chain need not be fixed at four steps, but may be expanded as required. For example, several additional sets of increasingly more specific and implementation dependent specifications may be included before the usable machine step. It has been suggested that the hierarchical decomposition used in the SRI method be included as a step in the chain, either before the formal specification step or immediately following it.

This method of validation offers the advantage that the number of steps in the chain can be made large enough that the correspondence becomes easy to understand. If each step is represented by a finite state automata, in order to demonstrate validation using this method, all that is required is to show that all of the state changes in one step correspond to all of the allowable state changes of the previous block.

8.1.4 Denning Method

D. E. Denning has proposed a different method from those discussed above. She has devised a method which is relatively independent of the methodology used to design the system and of the language used to implement it. As a basis, she has defined an information flow model and from this, she has derived a lattice to represent the secure model of an operating system. This method appears to be more mathematically rigorous and far easier to use than those previously described.

Most of the previous research has utilized the Bell and LaPadula model of a secure system as a starting point. D. E. Denning begins with an information flow model which appears to be far more general. With D.E. Denning's method, there is no restriction upon the methodology used to design the system. There are almost no restrictions upon the language used. Of the four mechanisms discussed, this is probably the easiest to implement. The Denning technique is described more fully in Volume VII.

8.2 SOFTWARE VERIFICATION TECHNIQUES

Once software has been written, it becomes necessary to establish that it does, indeed, function as intended. Volume VII explores the problems of showing that a program does function correctly, and of establishing the appropriate level of confidence in this demonstration. The first part of this analysis investigates the question of formally proving a program. Proving software has received a considerable amount of attention recently; its very name has inherent appeal. The second part of the discussion deals with the subject of symbolic execution. Symbolic execution is one of the tools used in both program testing and program proving, but is a tool with which few programmers appear to be familiar.

8.2.1 Program Proving

Program proving offers a tool for use in trying to establish the reliability of a program, but the tool is still weak. It does not as yet fulfill the extensive claims made for it. Research in this area has been continuing since 1967, when A. W. Floyd published his paper. The basic technique postulated by Floyd has received some elaboration, but it still remains basically unchanged; slow, cumbersome, uncertain and messy. With these deficiencies, it is not likely that any dramatic breakthroughs will be seen in the time frame applicable to CPOS development.

There is also a major difference between proofs of programs and mathematical proofs. When a mathematician successfully proves some new theorem, he publishes it in a journal. This journal is sent to hundreds or thousands of other mathematicians, some of whom will read and study this proof to decide whether it is of value in their research. If they find an error in the proof, the author will be quickly informed. Program proofs, on the other hand, will not receive this kind of scrutiny, in most cases, because they are of interest to virtually no one outside the project group. In other words, if a program proof contains an error, who is going to find it? One must consider the fact that program proving is at least as complex an intellectual activity as programming; therefore, a proof is as likely to contain an error as is the program being proven. In addition, because of the complexity of the program proof, it is probable that an error in the proof will not be found until a failure has occurred.

Program proving cannot be used to show conclusively that a program will perform as expected. There are simply too many variables, and in attempting to take them all into account, the proof of a relatively simple program can take several man-years to complete. Program proving techniques may prove to be useful, however, as a limited tool for detecting potential errors due to logical inconsistencies. This is especially true as attempts to have the computer generate the proofs become more successful.

Until research produces strong, automated proving systems, we believe that program proving is of limited value to the UDS effort and, therefore, recommend that formal proofs not be required.

8.2.2 Symbolic Execution

Symbolic execution is the process of using the computer to simulate the execution of a program using symbols in the place of actual data values. These symbols may represent any possible numeric value, or they may be constrained to a limited group of values, such as all values less than ten. Symbols are usually assigned as parameters by input statements or by the entry statement of the routine.

Symbolic execution has been used as a tool in several different lines of research in computer science. The most basic use is in testing programs which are under development. Another related line of research that it has been used in, is the generation of data set test cases. It has also been used as a tool in automated program proving, and as a tool for determining the specifications for existing, undocumented programs.

The authors believe that symbolic execution has been shown to be a viable, although still experimental, tool in the repertoire of the professional programmer. As with any tool, its use has both advantages and disadvantages, depending upon the way it is used. Three of the uses explored in Volume VII related to the CPOS design effort are program testing, data case generation, and program proving.

At present, many of the theorem proving routines require user assistance to manipulate the symbolic formulas for the variables and the path condition into the form of the exit assertions. Even more basic, most of the symbolic execution systems require user guidance in path selection, with loop structures being a major case in point. Any program with a multitude of internal paths threatens to cause the execution to run virtually endlessly, unless appropriate path guidance is provided. Before symbolic execution can be recommended for use in validating programs, extensive research in program proving is required, and it is doubtful that this research will produce meaningful results within the time frame of the UDS development effort.

The use of symbolic execution systems to generate test data cases has not been shown to be effective as evidenced by research using the DISSECT system. Volume VII explains DISSECT as well as several other systems. The basic problem is that a test data case only shows that the program will work for that test case. Generalizing a single case to show that the program will work for the input class of a program path is often difficult to demonstrate. Symbolic execution may be useful to the CPOS developers to generate test data during program development, but we cannot recommend this technique for final acceptance testing for these reasons.

As a tool for program development testing, symbolic execution offers a wealth of information to the programmer. It is particularly useful in solving the difficult problem of keeping track of the selection of the various alternate paths through the program. We recommend the use of symbolic execution in this application.

Plessey also recommends that the use of symbolic execution be included in the design of the higher level language processors to be used in coding the CPOS. The system should be batch oriented along the lines recommended by W.E. Howden, L.J. Osterweil, and L.D. Fosdick, which are discussed in Volume VII.

8.3 TESTING

Since program correctness proving does not appear to be a practical technique for validating the CPOS, we are left with the need for good program testing. In many past cases, testing of large software systems was incompletely performed or was excessively expensive. It was not unusual for a relatively simple routine to take as long to test and to debug as it took to design and code.

An error in a program can be corrected in the design phase relatively cheaply. Once coding has commenced, the cost begins to rise rapidly because of the recoding which will have to be performed. After the system has been integrated, correcting an error may create expensive cost and time overruns. If the software has been delivered, the correction not only involves money and man-hours, but also the secondary costs due to downtime and loss of service.

Because of the expense involved in testing, programming techniques have been developed which attempt to minimize both errors and testing. These techniques include top-down design, modularization or segmentation, design reviews, structured programming, program proving, and program simulation.

8.3.1 Test Plans

In order to test software effectively and to establish a level of confidence in the software, testing should not be done in an ad hoc manner. Unless testing proceeds according to some clearly defined plan, testing will likely be delayed until the very end, and will then be conducted in a rather haphazard manner. The completeness of testing will be a function of the time available, and that is usually insufficient. The results of an unplanned testing approach are systems that are unreliable.

The use of a test plan helps assure success during the testing phase. The test plan is a document which is prepared with the functional specifications for the software. If a top-down design approach is taken, a test plan should be written for each successive level of specification. As the software specifications grow and change, the test plans must also change.

The level of confidence which one has in a given piece of software is always somewhat subjective, and is affected by the thoroughness of testing and the probability of finding more latent errors in the future. These

items can, and should, be measured quantitatively and then be compared against some standard. This is the function of the reliability models discussed in Volume II.

8.3.2 Static Testing

Testing can be divided into two categories: static and dynamic. Static testing uses procedures which do not require execution of the code. This is usually the first phase of testing and can be divided into three parts: precompilation testing, compiler testing, and testing by a static program analyzer.

Precompilation testing consists of manual tests performed on the program code before an attempt is made to compile the code. This phase was more important in prior years when the cost of computer time was expensive. As the price of computers decrease and speed and efficiency increase, this type of testing is emphasized less.

Compiler testing is the next phase of the testing process. The compiler is a useful and powerful code testing tool. In order to produce executable machine code, a compiler provides syntactic and semantic analysis of the source code.

The rules of structured programming have been shown to lower the incidence of error. Two of these rules can be easily and inexpensively enforced by a compiler. These are the rules which restrict the use of the unconditional branch statement and which forbid the modification of a loop iteration control variable while inside the loop.

The static analyzer is designed to complement and extend the static testing performed by the compiler. Since there is no guarantee that a program has been successfully compiled before it is used as input to the static analyzer, there must be some overlap between the two testing tools. Testing with the static analyzer, however, is usually more rigorous.

The static analyzer can detect many types of errors relatively quickly and efficiently. The information revealed, especially by the static flow analysis, can simplify and shorten the time spent performing the more expensive dynamic testing.

8.3.3 Dynamic Testing

Some types of errors, particularly those sensitive to timing, are not revealed during static testing but require execution of the code for detection of errors. Dynamic testing is thus required. This involves establishing a suitable environment for the software under test and providing test data, either real or symbolic, while using some means of instrumentation to observe the software module's behavior. A number of different tools exist which can provide both the necessary environment and the means of instrumentation, and these are discussed in Volume VII. Dynamic testing can be subdivided into three phases: data generation, test run, and test output evaluation. Selecting the test data cases is, however, the most critical aspect of testing.

The testing criterion to be used in testing the CPOS must be sufficiently rigorous to execute each branch at least once. We do not believe that many CPOS routines will require testing of critical values by the techniques explained in Volume VII. Those that do, however, require special value testing in addition to path testing.

8.3.4 Debugging

Interactive debugging tools are often used to provide two distinct functions: program testing and program debugging. These terms are often used synonymously, but they have different meanings. Program testing is the process of locating errors, or "bugs," in a program. Program debugging is the process of correcting those errors. In a batch-oriented environment, the separation of these functions is enforced by the job-oriented nature of the system. The interactive environment allows the user to switch back and forth between testing and debugging at will: test, find an error, correct it, and test again. Interactive debuggers assist the programmer in making this transition by providing a number of useful features.

It is recommended that the debugger for the CPOS be implemented in a manner which allows the user to switch between interpretive execution and direct execution at will. The monitoring techniques described in Volume VII provide a necessary and useful window on the operation of the code being tested. It is recommended that as many as possible be included in the debug package for the CPOS.

8.3.5 Performance Testing

Performance testing is not designed to find coding errors as is the case for static and dynamic testing, but rather to measure efficiency, throughput, and error recovery performance.

Some performance monitoring probes are designed to provide "coarse-grained" data while others provide more detailed "fine-grained" data. Both types of monitoring probes are required for the CPOS. The data usually collected includes the frequency with which a process is created and destroyed, the procedures the process executes, the amount of time the process has use of a processor, the amount of time a process waits in various job queues, and what other resources the process utilizes. Sources of data for the monitors are generally system software probes, input/output device registers, specialized monitoring hardware, and specially designed software.

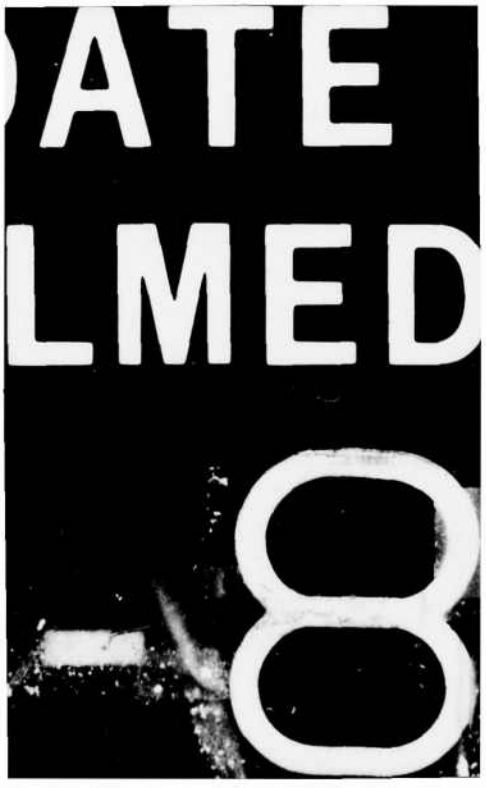
Other types of data can be gathered by the performance monitoring probes such as parameters passed between modules, calls to the security mechanism, page and segment faults, internal program paths execution, and DO loop activity.

The problem of obtaining statistically significant results is a challenging one. Event simulations are essentially statistical experiments, and it is often difficult to be sure in any particular case that the run has been long enough to be truly representative of the real world. Despite these difficulties, event simulators have proven to be indispensible tools in the design and evaluation of communications systems and, particularly, in the area of network and switch node modeling. A number of simulation models exist and are operated by the Air Force and the Defense Communications Agency. Event simulation data is useful in improving CPOS performance in terms of its capacity to handle its workload. This is particularly true as the implementors "tune" the design to maximize performance and efficiency.

Plessey recommends that on-line testing be built into the design of the CPOS. It is specifically recommended that boundary checks be applied to the indices involved in each matrix or table reference. The range of the parameters and returned values of each subprogram call should be checked, where practical. Further,they recommend that range testing be applied to entry points to and exit points from the CPOS and at the entry to and exit from critical sections within the CPOS. It is recommended that state checking be employed only if the hardware architecture design is not capable of enforcing the use of small domains of protection.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.


ATE
LMED
8